New Objects

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

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

Objects that have been added to Visual Basic in Microsoft Outlook 2002 are listed in the following table.

<table>
<thead>
<tr>
<th>Objects</th>
</tr>
</thead>
<tbody>
<tr>
<td>ItemProperties</td>
</tr>
<tr>
<td>ItemProperty</td>
</tr>
<tr>
<td>Reminder</td>
</tr>
<tr>
<td>Reminders</td>
</tr>
<tr>
<td>Results</td>
</tr>
<tr>
<td>Search</td>
</tr>
<tr>
<td>View</td>
</tr>
<tr>
<td>Views</td>
</tr>
</tbody>
</table>
New Events (Alphabetical)

Events that have been added to Visual Basic in Microsoft Outlook 2002 are listed in the following table (sorted alphabetically).

<table>
<thead>
<tr>
<th>Events</th>
</tr>
</thead>
<tbody>
<tr>
<td>AdvancedSearchComplete</td>
</tr>
<tr>
<td>AdvancedSearchStopped</td>
</tr>
<tr>
<td>BeforeDelete</td>
</tr>
<tr>
<td>BeforeItemCopy</td>
</tr>
<tr>
<td>BeforeItemCut</td>
</tr>
<tr>
<td>BeforeItemPaste</td>
</tr>
<tr>
<td>BeforeMaximize</td>
</tr>
<tr>
<td>BeforeMinimize</td>
</tr>
<tr>
<td>BeforeMove</td>
</tr>
<tr>
<td>BeforeReminderShow</td>
</tr>
<tr>
<td>BeforeSize</td>
</tr>
<tr>
<td>MapiLogonComplete</td>
</tr>
<tr>
<td>ReminderAdd</td>
</tr>
<tr>
<td>ReminderChange</td>
</tr>
<tr>
<td>ReminderFire</td>
</tr>
<tr>
<td>ReminderRemove</td>
</tr>
<tr>
<td>Snooze</td>
</tr>
<tr>
<td>ViewAdd</td>
</tr>
<tr>
<td>ViewRemove</td>
</tr>
</tbody>
</table>
New Events (by Object)

Events that have been added to Visual Basic in Microsoft Outlook 2002 are listed in the following table (sorted by object name).

<table>
<thead>
<tr>
<th>Objects</th>
<th>Events</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application</td>
<td>AdvancedSearchComplete</td>
</tr>
<tr>
<td></td>
<td>AdvancedSearchStopped</td>
</tr>
<tr>
<td></td>
<td>MapiLogonComplete</td>
</tr>
<tr>
<td>AppointmentItem</td>
<td>BeforeDelete</td>
</tr>
<tr>
<td>ContactItem</td>
<td>BeforeDelete</td>
</tr>
<tr>
<td>DistListItem</td>
<td>BeforeDelete</td>
</tr>
<tr>
<td>DocumentItem</td>
<td>BeforeDelete</td>
</tr>
<tr>
<td>Explorer</td>
<td>BeforeMaximize</td>
</tr>
<tr>
<td></td>
<td>BeforeMinimize</td>
</tr>
<tr>
<td></td>
<td>BeforeMove</td>
</tr>
<tr>
<td></td>
<td>BeforeSize</td>
</tr>
<tr>
<td>Inspector</td>
<td>BeforeMaximize</td>
</tr>
<tr>
<td></td>
<td>BeforeMinimize</td>
</tr>
<tr>
<td></td>
<td>BeforeMove</td>
</tr>
<tr>
<td></td>
<td>BeforeSize</td>
</tr>
<tr>
<td>JournalItem</td>
<td>BeforeDelete</td>
</tr>
<tr>
<td>MailItem</td>
<td>BeforeDelete</td>
</tr>
<tr>
<td>MeetingItem</td>
<td>BeforeDelete</td>
</tr>
<tr>
<td>PostItem</td>
<td>BeforeDelete</td>
</tr>
<tr>
<td>Reminders</td>
<td>BeforeReminderShow</td>
</tr>
<tr>
<td></td>
<td>ReminderAdd</td>
</tr>
<tr>
<td></td>
<td>ReminderChange</td>
</tr>
<tr>
<td></td>
<td>ReminderFire</td>
</tr>
<tr>
<td>Category</td>
<td>Action</td>
</tr>
<tr>
<td>---------------------</td>
<td>----------------</td>
</tr>
<tr>
<td>Reminder</td>
<td>Remove</td>
</tr>
<tr>
<td>RemoteItem</td>
<td>BeforeDelete</td>
</tr>
<tr>
<td>ReportItem</td>
<td>BeforeDelete</td>
</tr>
<tr>
<td>TaskItem</td>
<td>BeforeDelete</td>
</tr>
<tr>
<td>TaskRequestAcceptItem</td>
<td>BeforeDelete</td>
</tr>
<tr>
<td>TaskRequestDeclineItem</td>
<td>BeforeDelete</td>
</tr>
<tr>
<td>TaskRequestItem</td>
<td>BeforeDelete</td>
</tr>
<tr>
<td>TaskRequestUpdateItem</td>
<td>BeforeDelete</td>
</tr>
<tr>
<td>Views</td>
<td>ViewAdd</td>
</tr>
<tr>
<td></td>
<td>ViewRemove</td>
</tr>
</tbody>
</table>
New Methods (Alphabetical)

Methods that have been added to Visual Basic in Microsoft Outlook 2002 are listed in the following table (sorted alphabetically).

<table>
<thead>
<tr>
<th>Methods</th>
</tr>
</thead>
<tbody>
<tr>
<td>AddMember</td>
</tr>
<tr>
<td>AddToFavorites</td>
</tr>
<tr>
<td>AddToPFFavorites</td>
</tr>
<tr>
<td>AdvancedSearch</td>
</tr>
<tr>
<td>CopyFile</td>
</tr>
<tr>
<td>Dial</td>
</tr>
<tr>
<td>Dismiss</td>
</tr>
<tr>
<td>GoToDate</td>
</tr>
<tr>
<td>IsSearchSynchronous</td>
</tr>
<tr>
<td>RemoveMember</td>
</tr>
<tr>
<td>RemoveStore</td>
</tr>
<tr>
<td>Reset</td>
</tr>
<tr>
<td>SetIcon</td>
</tr>
<tr>
<td>ShowCategoriesDialog</td>
</tr>
</tbody>
</table>
New Methods (by Object)

Methods that have been added in to Visual Basic Microsoft Outlook 2002 are listed in the following table (sorted by object name).

<table>
<thead>
<tr>
<th>Objects</th>
<th>Methods</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application</td>
<td>AdvancedSearch</td>
</tr>
<tr>
<td></td>
<td>CopyFile</td>
</tr>
<tr>
<td></td>
<td>IsSearchSynchronous</td>
</tr>
<tr>
<td>AppointmentItem</td>
<td>ShowCategoriesDialog</td>
</tr>
<tr>
<td>ContactItem</td>
<td>ShowCategoriesDialog</td>
</tr>
<tr>
<td>DistListItem</td>
<td>AddMember</td>
</tr>
<tr>
<td></td>
<td>RemoveMember</td>
</tr>
<tr>
<td></td>
<td>ShowCategoriesDialog</td>
</tr>
<tr>
<td>DocumentItem</td>
<td>ShowCategoriesDialog</td>
</tr>
<tr>
<td>JournalItem</td>
<td>ShowCategoriesDialog</td>
</tr>
<tr>
<td>MailItem</td>
<td>ShowCategoriesDialog</td>
</tr>
<tr>
<td>MAPIFolder</td>
<td>AddToFavorites</td>
</tr>
<tr>
<td></td>
<td>AddToPFFavorites</td>
</tr>
<tr>
<td>MeetingItem</td>
<td>ShowCategoriesDialog</td>
</tr>
<tr>
<td>NameSpace</td>
<td>Dial</td>
</tr>
<tr>
<td></td>
<td>RemoveStore</td>
</tr>
<tr>
<td>OutlookBarShortcut</td>
<td>SetIcon</td>
</tr>
<tr>
<td>PostItem</td>
<td>ShowCategoriesDialog</td>
</tr>
<tr>
<td>Reminder</td>
<td>Dismiss</td>
</tr>
<tr>
<td>RemoteItem</td>
<td>ShowCategoriesDialog</td>
</tr>
<tr>
<td>ReportItem</td>
<td>ShowCategoriesDialog</td>
</tr>
<tr>
<td>TaskItem</td>
<td>ShowCategoriesDialog</td>
</tr>
<tr>
<td>TaskRequestAcceptItem</td>
<td>ShowCategoriesDialog</td>
</tr>
<tr>
<td>TaskRequestDeclineItem</td>
<td>ShowCategoriesDialog</td>
</tr>
<tr>
<td>TaskRequestItem</td>
<td>ShowCategoriesDialog</td>
</tr>
<tr>
<td>TaskRequestUpdateItem</td>
<td>ShowCategoriesDialog</td>
</tr>
<tr>
<td>----------------------</td>
<td>-----------------------</td>
</tr>
<tr>
<td>View</td>
<td>GoToDate</td>
</tr>
<tr>
<td></td>
<td>Reset</td>
</tr>
</tbody>
</table>
# New Properties (Alphabetical)

Properties that have been added to Visual Basic in Microsoft Outlook 2002 are listed in the following table (sorted alphabetically).

<table>
<thead>
<tr>
<th>Properties</th>
</tr>
</thead>
<tbody>
<tr>
<td>AddressBookName</td>
</tr>
<tr>
<td>AppFolders</td>
</tr>
<tr>
<td>BodyFormat</td>
</tr>
<tr>
<td>CustomViewsOnly</td>
</tr>
<tr>
<td>DownloadState</td>
</tr>
<tr>
<td>Filter</td>
</tr>
<tr>
<td>FolderPath</td>
</tr>
<tr>
<td>HTMLDocument</td>
</tr>
<tr>
<td>IMAddress</td>
</tr>
<tr>
<td>InAppFolderSyncObject</td>
</tr>
<tr>
<td>InternetCodepage</td>
</tr>
<tr>
<td>IsConflict</td>
</tr>
<tr>
<td>IsSynchronous</td>
</tr>
<tr>
<td>IsUserProperty</td>
</tr>
<tr>
<td>IsVisible</td>
</tr>
<tr>
<td>ItemProperties</td>
</tr>
<tr>
<td>LastFirstNoSpaceAndSuffix</td>
</tr>
<tr>
<td>LockUserChanges</td>
</tr>
<tr>
<td>MarkForDownload</td>
</tr>
<tr>
<td>NextReminderDate</td>
</tr>
<tr>
<td>Offline</td>
</tr>
<tr>
<td>OriginalReminderDate</td>
</tr>
<tr>
<td>Reminders</td>
</tr>
<tr>
<td>Results</td>
</tr>
</tbody>
</table>
SaveOption
Scope
SearchSubFolders
ShowAsOutlookAB
Tag
Views
XML
## New Properties (by Object)

Properties that have been added to Visual Basic in Microsoft Outlook 2002 are listed in the following table (sorted by object name).

<table>
<thead>
<tr>
<th>Objects</th>
<th>Properties</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application</td>
<td>Reminders</td>
</tr>
<tr>
<td></td>
<td>DownloadState</td>
</tr>
<tr>
<td></td>
<td>InternetCodepage</td>
</tr>
<tr>
<td>AppointmentItem</td>
<td>IsConflict</td>
</tr>
<tr>
<td></td>
<td>ItemProperties</td>
</tr>
<tr>
<td></td>
<td>MarkForDownload</td>
</tr>
<tr>
<td></td>
<td>DownloadState</td>
</tr>
<tr>
<td></td>
<td>IMAddress</td>
</tr>
<tr>
<td></td>
<td>IsConflict</td>
</tr>
<tr>
<td>ContactItem</td>
<td>ItemProperties</td>
</tr>
<tr>
<td></td>
<td>LastFirstNoSpaceAndSuffix</td>
</tr>
<tr>
<td>DistListItem</td>
<td>ItemProperties</td>
</tr>
<tr>
<td></td>
<td>MarkForDownload</td>
</tr>
<tr>
<td>DocumentItem</td>
<td>ItemProperties</td>
</tr>
<tr>
<td></td>
<td>MarkForDownload</td>
</tr>
<tr>
<td></td>
<td>DownloadState</td>
</tr>
<tr>
<td>Exception</td>
<td>ItemProperties</td>
</tr>
<tr>
<td>Explorer</td>
<td>HTMLDocument</td>
</tr>
<tr>
<td>ItemProperty</td>
<td>Views</td>
</tr>
<tr>
<td>JournalItem</td>
<td>IsUserProperty</td>
</tr>
<tr>
<td></td>
<td>DownloadState</td>
</tr>
<tr>
<td></td>
<td>IsConflict</td>
</tr>
<tr>
<td></td>
<td>ItemProperties</td>
</tr>
</tbody>
</table>
**MailItem**
- MarkForDownload
- BodyFormat
- DownloadState
- InternetCodepage
- IsConflict
- ItemProperties
- MarkForDownload

**MAPIFolder**
- AddressBookName
- CustomViewsOnly
- FolderPath
- InAppFolderSyncObject
- ShowAsOutlookAB
- Views
- DownloadState
- IsConflict

**MeetingItem**
- ItemProperties
- MarkForDownload

**NameSpace**
- Offline
- DownloadState

**NoteItem**
- ItemProperties
- MarkForDownload
- BodyFormat
- DownloadState

**PostItem**
- InternetCodepage
- IsConflict
- ItemProperties
- MarkForDownload

**Reminder**
- IsVisible
- NextReminderDate
- OriginalReminderDate
- DownloadState
- IsConflict

**RemoteItem**
- ItemProperties
- MarkForDownload
- DownloadState
- IsConflict
<table>
<thead>
<tr>
<th>Table Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ReportItem</td>
<td>ItemProperties MarkForDownload</td>
</tr>
<tr>
<td>Search</td>
<td>Filter IsSynchronous Results Scope SearchSubFolders Tag</td>
</tr>
<tr>
<td>SyncObjects</td>
<td>AppFolders DownloadState InternetCodepage</td>
</tr>
<tr>
<td>TaskItem</td>
<td>IsConflict ItemProperties MarkForDownload DownloadState</td>
</tr>
<tr>
<td>TaskRequestAcceptItem</td>
<td>IsConflict ItemProperties MarkForDownload DownloadState</td>
</tr>
<tr>
<td>TaskRequestDeclineItem</td>
<td>IsConflict ItemProperties MarkForDownload DownloadState</td>
</tr>
<tr>
<td>TaskRequestItem</td>
<td>IsConflict ItemProperties MarkForDownload DownloadState</td>
</tr>
<tr>
<td>TaskRequestUpdateItem</td>
<td>IsConflict ItemProperties MarkForDownload DownloadState</td>
</tr>
<tr>
<td>UserProperty</td>
<td>IsUserProperty LockUserChanges</td>
</tr>
<tr>
<td>View</td>
<td>SaveOption XML</td>
</tr>
</tbody>
</table>
Using Visual Basic for Applications in Outlook

Visual Basic for Applications in Microsoft Outlook makes it easy to control Outlook within Outlook itself. Using Visual Basic for Applications in Outlook, you can create macros that perform complex or repetitive tasks automatically. You can also develop program code that responds to Outlook events, allowing you to automate common tasks (such as arranging windows when Outlook starts).

Visual Basic for Applications in Outlook allows you to take full advantage of the Outlook object model, including the wide range of application-level events, without requiring you to run an external application (such as another Microsoft Office application or an application developed using Microsoft Visual Basic). And unlike form scripts developed using VBScript, Outlook Visual Basic for Applications code is always available in the application; an item does not have to be open to run the code.

All Outlook Visual Basic for Applications code is contained in a project. The project is associated with a particular user, so all users who run Outlook on a computer can customize Outlook to meet their own needs. A project can contain code modules and User Form modules (note that User Form modules are not the same as Outlook forms).

You use the Visual Basic Editor to create and remove modules, to design User Form modules, and to edit code in modules. This editor provides a powerful set of tools, including a built-in Object Browser and debugger to make developing and troubleshooting code easy. You can even use the Visual Basic Editor in Outlook to develop and test code that you can then copy to a standalone Visual Basic application or a Visual Basic for Applications application in another Microsoft Office application.

You can export a module to a file; this makes it easy to share your programs with other users, who can use the Visual Basic Editor to import the module into their
own Outlook Visual Basic for Applications projects.

Learn about writing an Outlook macro, responding to Outlook events, and designing Visual Basic for Applications forms.
Writing an Outlook macro

A macro is any public subroutine in a code module. A function or a private subroutine cannot be a macro, and a macro cannot be located in a class or form module.

To create a new macro

1. In Outlook, point to **Macro** on the **Tools** menu, and then click **Visual Basic Editor**.
2. In the **Project** window, double-click the module you want to contain the macro.
3. On the **Insert** menu, click **Procedure**.
4. In the **Name** box, type a name for the macro. The name cannot contain spaces.
5. Click **OK**.

   The template for the macro subroutine appears in the code module window.

6. Type the code you want to run in the body of the subroutine.

For more information about using the Visual Basic Editor, see the Visual Basic Editor Help.

Once you’ve created a macro, you can create a menu item or toolbar button that will run the macro when you click it.
Using Outlook Visual Basic for Applications to respond to Outlook events

You write an event procedure (also known as an event handler) to respond to events that occur in Microsoft Outlook. For example, you can write an event procedure that automatically maximizes the explorer window when Outlook starts.

Events are associated with particular objects. The Application object is the topmost object, and is always available (that is, it does not have to be created). You can add an Application event procedure in the ThisOutlookSession module window simply by selecting Application in the left list and then selecting the event in the right list.

Adding an event handler for objects other than the Application object requires a few additional steps.

First, you must declare a variable using the WithEvents keyword to identify the object whose event you want to handle. For example, to declare a variable representing the OutlookBarPane object, you would add the following to a code module.

Dim WithEvents myOlBar As Outlook.OutlookBarPane

You can then select myOlBar in the Objects list of the module window and then select the event in the procedure list. The Visual Basic Editor will then add the template for the event procedure to the module window. You can then type the code you want to run when the event occurs. The following example shows code added to the BeforeNavigate event procedure for the OutlookBarPane object.

Private Sub myOlBar_BeforeNavigate(ByVal Shortcut As OutlookBarShortcut, Cancel As Boolean)
    If Shortcut.Name = "Notes" Then
        MsgBox "You cannot open the Notes folder."
    End If
End Sub
The final step is to add code to set the object variable to the object whose event you want to handle. This code can exist in a macro, or if you want the event to be handled whenever Outlook runs, you can put it in the **Startup** event procedure, as in the following example.

```vba
Private Sub Application_Startup()
    Set myOlBar = Application.ActiveExplorer.Panes(1)
End Sub
```
Working with forms in the Visual Basic Editor

You can use the Visual Basic Editor to design a form that allows your users to interact with your Microsoft Visual Basic for Applications program. Unlike an Outlook form, a Visual Basic for Applications form is not used to display an Outlook item, nor can a control on a Visual Basic for Applications form be bound to an item field.

Your Visual Basic for Applications program can use a Visual Basic for Applications user form to gather information from your users; your program can then use this information to set properties of new or existing Outlook items. For example, a program that creates a boilerplate mail message could use a Visual Basic for Applications form to allow the user to enter the specific information for the message to be sent. When the user closes the form, the program uses the information in the form to set the properties of the mail message and then sends the message.

The following sample uses the text in two text boxes to add information to a message before sending it.

```vba
Private Sub CommandButton1_Click()
    Dim myMail As Outlook.MailItem
    Set myMail = Application.CreateItem(olMailItem)
    With myMail
        .To = TextBox1.Text
        .Subject = "Book overdue: " & TextBox2.Text
        .Body = "Please return this book as soon as possible."
    End With
    myMail.Send
End Sub
```

You can also use controls to display information about Outlook items, folders, and other features of the Outlook object model. The following example shows how to fill a combo box control with the subjects of the items in the user’s Inbox.
Dim myItems As Outlook.Items
Set myItems = Application.GetNamespace("MAPI").GetDefaultFolder(olFolderInbox).Items
For x = 1 To myItems.Count
    ComboBox1.AddItem myItems.Item(x).Subject
Next x

For more information about creating and using forms in the Visual Basic Editor, see the Visual Basic Editor Help.
Automating Outlook from a Visual Basic Applications

Because Microsoft Outlook supports Automation, you can control Outlook from any program written with Microsoft Visual Basic. Automation provides a standard method for one application to access the objects, methods, properties, and events of other applications that support Automation.

The Outlook object model provides all of the functionality necessary to manipulate data stored in Outlook folders, and it provides the ability to control many aspects of the Outlook user interface.

To start an Outlook automation session, you can use either early or late binding. Late binding uses either the GetObject or the CreateObject function to initialize Outlook. For example, the following code sets an object variable to the Outlook Application object, which is the highest-level object in the Outlook object model. All automation code must first define an Outlook Application object to be able to access any other Outlook objects.

Dim objOL as Object
Set objOL = CreateObject("Outlook.Application")

To use early binding, you first need to set a reference to the Outlook object library. You can then use the following syntax to start an Outlook session.

Dim objOL as Outlook.Application
Set objOL = New Outlook.Application

Most programming solutions interact with the data stored in Outlook. Outlook stores all of its information in Messaging Application Programming Interface (MAPI) folders. After you set an object variable to the Outlook Application object, you will commonly set a Namespace object to refer to MAPI, as shown in the following example.

Set objOL = New Outlook.Application
Set objNS = objOL.GetNameSpace("MAPI")
Set objFolder = objNS.GetDefaultFolder(olFolderContacts)

Once you have set an object variable to reference the folder that contains the items you wish to work with, you use appropriate code to accomplish your task, as shown in the following example.

Sub CreateNewDefaultOutlookTask()
    Dim objOLApp As Outlook.Application
    Dim NewTask As Outlook.TaskItem
    ' Set the Application object
    Set objOLApp = New Outlook.Application
    ' You can only use CreateItem for default items
    Set NewTask = objOLApp.CreateItem(olTaskItem)
    ' Display the new task form so the user can fill it out
    NewTask.Display
End Sub
Automating Outlook from other Microsoft Office applications

You can use Microsoft Visual Basic for Applications in any Microsoft Office application to control Microsoft Outlook. For example, if you are developing a cross-application solution using one primary application and several secondary applications, you can write Visual Basic for Applications code in the primary application to automate Outlook to send messages and to store and retrieve information in Outlook items. For example, in Microsoft Excel you can write routines that send a workbook to an Outlook distribution list.

To control Outlook objects from outside Outlook, you must establish a reference to the Outlook object library from the project in which you are writing code. To do this, use the References dialog box in the Visual Basic Editor in the primary application. You can then write code that returns a reference to the Outlook Application object. Through this reference, your code has access to all the objects, properties, methods, and constants defined in the Outlook type library.

There are several ways to return a reference to the Outlook Application object:

- You can use the CreateObject function to start a new session of Outlook and return a reference to the Application object that represents the new session.
- You can use the GetObject function to return a reference to the Application object that represents a session that’s already running. Note that because there can be only one instance of Outlook running at any given time, GetObject usually serves little purpose when used with Outlook. CreateObject can always be used to access the current instance of Outlook or to create a new instance if one does not exist. However, you can use error trapping with the GetObject method to determine if Outlook is currently running.
- You can use the New keyword in several types of statements to implicitly create a new instance of the Outlook Application object using the Set statement to set an object variable to the new instance of the Application
object. You can also use the New keyword with the Dim, Private, Public, or Static statement to declare an object variable. The new instance of the Application object is then created on the first reference to the variable.

Automating Outlook from a Visual Basic Application provides examples of using these methods of referencing the Outlook Application object.
Working with Outlook events

Microsoft Outlook provides a wide range of events through which it can notify your Microsoft Visual Basic, Microsoft Visual Basic for Applications, and VBScript programs that a significant change has occurred. For example, Outlook events can notify a program when a new item has been opened or that the user has changed the Outlook Bar.

To receive notification of a significant event, write an event-handler procedure. Depending on whether the event is handled in Visual Basic or Visual Basic for Applications or in VBScript, this is either a Sub or a Function that Outlook calls when the event is called. The code you put in the event handler allows your program to respond appropriately to the event and, in some cases, even lets your program cancel the default action associated with the event, such as preventing a mail item from being sent.
Types of Events

Outlook events can be divided into two main categories: item-level events and application-level events.

Item-level events pertain to a particular item, and are typically handled by VBScript code contained within the form associated with the item. These events notify your program when an item has been opened, sent or posted, saved, or closed, and when the user has replied to or forwarded a message or initiated a custom action. Item-level events can also notify your program when the user has clicked a control on the form or when an item property has changed.

Application-level events are typically handled by Visual Basic or Visual Basic for Applications because they pertain to more than the items associated with a particular form. Application-level events can pertain to the application itself, to explorer collections and windows (including the Outlook Bar), inspector collections and windows, folders and folders collections, items collections, and synchronization objects.
Responding to Events

To respond to item-level events, add event-handler procedures to the script of the form that displays the item. For example, to run code when an item is opened in the form, add a procedure like the following to the script in the form.

```vba
Function Item_Open()
    MsgBox "A new item has opened in this form."
End Function
```

Responding to application-level events is somewhat more involved because steps must be taken to associate the event handler with the part of Outlook in which the event is occurring. Learn about writing an application-level event handler.
Order of Events

Except for certain form events, your program cannot assume that events will occur in a particular order, even if they appear to be called in a consistent sequence. The order in which Outlook calls event handlers might change depending on other events that might occur, or the order might change in future versions of Outlook.
Customizing Outlook using COM add-ins

You can use Microsoft Visual Basic version 5.0 or later (version 6.0 or later is preferred) or the Microsoft Office Developer to create a COM add-in to extend and enhance Microsoft Outlook.

Creating a COM add-in involves two major steps:

1. Implement the IDTExtensibility2 interface in a class module of a dynamic link library (DLL).
2. Register the COM add-in.
Implement the IDTExtensibility2 interface

The IDTExtensibility2 interface consists of five event procedures. To implement this interface in a Visual Basic program, set a reference to the Microsoft Add-In Designer object library and then add the following statement to the Declarations section of a class module:

`Implements IDTExtensibility2`

You can then add the empty event procedures to the code window of the class module and add your own program code to the procedures. You can also copy the empty procedures from an Outlook COM Add-in Template.
Register the COM add-in

In order to work with Outlook, the add-in DLL must be registered. The DLL’s class ID is registered beneath the \HKEY_CLASSES_ROOT subtree in the registry.

In addition, information about the add-in must be added to the registry. This information provides the add-in’s name, description, target application, initial load behavior, and connection state.

**Note** If you use Microsoft Visual Basic 6.0 or later Developer to design your COM add-in, the add-in designer will perform the steps required to register the COM add-in for you.

The following example shows the contents of a sample registry-editor (.reg) file that illustrates how to register an Outlook COM add-in.

```
[HKEY_CURRENT_USER\Software\Microsoft\Office\Outlook\Addins\SampleAddIn.AddInIFace]
"FriendlyName"="Sample Add-in"
"Description"="Sample Outlook Add-In"
"LoadBehavior"=dword:00000008
```

When the COM add-in is first registered, **LoadBehavior** can be set to any of the following flags.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>Load at startup. The COM add-in is to be loaded and connected when Outlook starts.</td>
</tr>
<tr>
<td>8</td>
<td>Load on demand. The COM add-in is to be loaded and connected only when the user requests it, such as by using the <strong>COM Add-ins</strong> dialog box.</td>
</tr>
<tr>
<td>16</td>
<td>Connect first time. The COM add-in is loaded and connected the first time the user runs Outlook after the COM add-in has been registered. The next time Outlook is run, the COM add-in is loaded when the user requests it. Use this value if your COM add-in modifies the user interface to allow the user to request the COM add-in be connected on demand (by clicking a button, for</td>
</tr>
</tbody>
</table>
After the COM add-in is registered and loaded, the LoadBehavior value can be combined with either of the following two flags to indicate current connection state of the COM add-in.

**Flag  Description**

<table>
<thead>
<tr>
<th>Flag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Disconnected</td>
</tr>
<tr>
<td>1</td>
<td>Connected</td>
</tr>
</tbody>
</table>

To connect the COM add-in, set the Connected flag in LoadBehavior; clear the flag to disconnect the COM add-in.

The FriendlyName value specifies the name of the COM add-in as it’s displayed in the COM Add-in dialog box. The Description value provides additional information about the COM add-in.
Adding custom property pages

Creating a custom property page for Microsoft Outlook involves four major steps:

1. Create the page as an ActiveX control.
2. Implement the PropertyPage object.
3. Write a procedure that sets the value of the Dirty property and calls the OnStatusChange method.
4. Create a Component Object Model (COM) add-in that contains an event procedure for OptionsPagesAdd.
Create the page as an ActiveX control

A custom property page in Outlook is an ActiveX control that’s implemented along with a dynamic link library (DLL) that’s designed as a COM add-in. The easiest way to create a custom property page is using Microsoft Visual Basic version 6.0 or higher. This version of Visual Basic provides templates and tools that simplify the process of creating both ActiveX controls and COM add-ins.

When you create the ActiveX control, you populate it with the controls your user will require to set the properties your page is designed to support. Because Outlook might resize the control when it displays the property page in the dialog box, the control’s Initialize event should position and size the child controls dynamically, depending on the final values of the control’s Width and Height properties.

The dialog box in which the custom property page is displayed has three buttons below the property pages: an OK button, a Cancel button, and an Apply button. When the user clicks the OK button, changes to properties on all pages in the dialog box are applied and the dialog box is closed. If the user clicks the Cancel button, no properties are changed and the dialog box is closed. If the user clicks the Apply button, any changes to properties are applied but the dialog box remains open. You should design your property page to respond appropriately when the user clicks these buttons. Later sections in this topic describe how to notify Outlook that the status of your property page has changed and how Outlook notifies your program when the changed property values should be applied.

Implement the PropertyPage object

The PropertyPage object is an abstract object; that is, its interfaces are defined but not implemented by Outlook. If your custom property change will rely on the Apply or Help button of the parent dialog box, the module that contains the custom property page ActiveX control must implement the PropertyPage object. To implement the object, the module must have a reference set to the Microsoft Outlook 9 Object Library and must contain the following statement.

Implements Outlook.PropertyPage
The module must then contain code that implements the methods and properties of the **PropertyPage** object. The following table describes these procedures.

<table>
<thead>
<tr>
<th>Procedure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Dirty</strong> property</td>
<td>Called by Outlook in response to the <strong>OnStatusChange</strong> method to determine whether the user has changed a value on the property page. <strong>Apply</strong> method Called by Outlook to notify your program that the user has clicked the <strong>OK</strong> or the <strong>Apply</strong> button. Usually this procedure applies any property values changed by the user in the property page. <strong>GetPageInfo</strong> method Called by Outlook to obtain the Help file and topic associated with the property page.</td>
</tr>
</tbody>
</table>
Write a procedure that sets the Dirty property and calls the OnStatusChange method

Most commonly, changes to property values are not applied immediately in response to user interaction with the controls that let the user specify those values. Instead, the values are applied when the user clicks OK or Apply on the dialog box. The Apply button is grayed until the user changes a value on a property page. To notify Outlook that the user has changed a value on your property page, your program should call the OnStatusChange method and then return True when Outlook queries the Dirty property.
Create a COM add-in containing an event procedure for the OptionsPagesAdd event

The **OptionsPagesAdd** event gives your program the opportunity to add your custom property page to the Microsoft Outlook Options dialog box (if the event is fired for the **Application** object) or the folders Properties dialog box (if the event is called for the **NameSpace** object). When Outlook calls this event procedure, it passes a **PropertyPages** object. Your event procedure uses the **Add** method of the collection to add the **PropertyPage** object implemented by your program to the object.
**Action Object**

<table>
<thead>
<tr>
<th>Actions</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>NameSpace</td>
<td></td>
</tr>
</tbody>
</table>

Represents a specialized action (for example, the voting options response) that can be executed on an item. The **Action** object is a member of the **Actions** object.
Using the Action Object

Use **Actions**(index), where index is the name of an available action, to return a single **Action** object.

The following Visual Basic for Applications example uses the Reply action of a particular item to send a reply.

Set myOlApp = CreateObject("Outlook.Application")
myItem = CreateItem(olMailItem)
Set myReply = myItem.Actions("Reply").Execute

The following Visual Basic for Applications example does the same thing, using a different reply style for the reply.

Set myOlApp = CreateObject("Outlook.Application")
myItem = CreateItem(olMailItem)
myItem.Actions("Reply").ReplyStyle = _
    olIncludeOriginalText
Set myReply = myItem.Actions("Reply").Execute
Show All
Actions Object

Multiple objects

A collection of Action objects that represent all the specialized actions that can be executed on an Outlook item.
Using the Actions Object

Use the **Actions** property to return the **Actions** object for any Outlook item object.

Use **Actions(index)**, where *index* is the name of an available action, to return a single **Action** object.

The following Visual Basic for Applications example uses the Reply action of a particular item to send a reply.

```vba
Set myOlApp = CreateObject("Outlook.Application")
myItem = CreateItem(olMailItem)
Set myReply = myItem.Actions("Reply").Execute
```
AddressEntries Object

The AddressEntries collection is a collection of addresses in an AddressList object. The object may contain zero or more AddressEntry objects and provides access to the entries in a transport provider's address book container.
Using the AddressEntries Object

The following example sets a reference to an AddressEntries object.

Set myOlApp = CreateObject("Outlook.Application")
Set myNameSpace = myOlApp.GetNameSpace("MAPI")
Set myAddressList = myNameSpace.AddressLists("Personal Address Book")
Set myAddressEntries = myAddressList.AddressEntries

You can also index directly into the AddressEntries object, returning an AddressEntry object.

Set myAddressEntry = myAddressList.AddressEntries(index)
AddressEntry Object

Multiple objects

The AddressEntry object is an address in an AddressEntries object. Each AddressEntry object in the AddressEntries object holds information that represents a person or process to which the messaging system can deliver messages.
Using the AddressEntry Object

The following example sets a reference to an AddressEntry object.

Set myAddressEntry = myRecipient.AddressEntry

Use AddressEntries(index), where index is the index number of an address entry or a value used to match the default property of an address entry, to return a single AddressEntry object.

Set myAddressEntry = myAddressEntries.Item(index)
AddressList Object

The **AddressList** object is an address book that contains a set of **AddressEntry** objects. For instance, the Personal Address List is an **AddressList** object, as shown in the following example:

Set myAddressList = Application.Session.AddressLists("Personal Addre
Using the AddressList Object

The **AddressList** object supplies a list of address entries to which a messaging system can deliver messages. An **AddressList** object represents one address book container available under the transport provider's address book hierarchy for the current **session**. The entire hierarchy is available through the parent **AddressLists** object.
Show All
AddressLists Object

The AddressLists object contains a set of AddressList objects. The AddressLists collection provides access to the root of the transport provider's address book hierarchy for the current session.
Using the AddressLists Object

The following example sets a reference to the AddressLists object.

Set myAddressLists = myNameSpace.AddressLists
Application Object

Represented by the 

Application Multiple objects

This is the only object in the hierarchy that can be returned by using the CreateObject method or the intrinsic Visual Basic GetObject function.

The Outlook Application object has several purposes:

- As the root object, it allows access to other objects in the Outlook hierarchy.
- It allows direct access to a new item created by using CreateItem, without having to traverse the object hierarchy.
- It allows access to the active interface objects (the explorer and the inspector).
Using the Application Object

When you use Automation to control Microsoft Outlook from another application, you use the **CreateObject** method to create an Outlook **Application** object.

The following Visual Basic for Application example starts Microsoft Outlook (if it's not already running) and opens the default Inbox folder.

```vba
Set myOlApp = CreateObject("Outlook.Application")
Set myNameSpace = myOlApp.GetNameSpace("MAPI")
Set myFolder = myNameSpace.GetDefaultFolder(olFolderInbox)
myFolder.Display
```

The following Visual Basic for Applications example uses the **Application** object to create and open a new contact.

```vba
Set myOlApp = CreateObject("Outlook.Application")
Set myItem = myOlApp.CreateItem(olContactItem)
myItem.Display
```
AppointmentItem Object

Multiple objects AppointmentItem

 Represents an appointment in the Calendar folder. An AppointmentItem object can represent a meeting, a one-time appointment, or a recurring appointment or meeting.
Using the AppointmentItem Object

Use the `CreateItem` method to create an `AppointmentItem` object that represents a new appointment.

The following Visual Basic for Applications example returns a new appointment.

```vba
Set myOlApp = CreateObject("Outlook.Application")
Set myItem = myOlApp.CreateItem(olAppointmentItem)
```

Use `Items(index)`, where `index` is the index number of an appointment or a value used to match the default property of an appointment, to return a single `AppointmentItem` object from a Calendar folder.

You can also return an `AppointmentItem` object from a `MeetingItem` object by using the `GetAssociatedAppointment` method.
Attachment Object

Represents a document or link to a document contained in an Outlook item.
Using the Attachment Object

Use **Attachments(index)**, where *index* is the index number, to return a single **Attachment** object.

Use the **Add** method to add an attachment to an item.

The following Visual Basic for Applications example creates a new mail message, attaches Q496.xls as an attachment (not a link), and gives the attachment a descriptive caption.

```vbnet
Set myOlApp = CreateObject("Outlook.Application")
Set myItem = myOlApp.CreateItem(olMailItem)
Set myAttachments = myItem.Attachments
myAttachments.Add "C:\My Documents\Q496.xls", _
               olByValue, 1, "4th Quarter 1996 Results Chart"
```
Attachments Object

Multiple objects

An object containing Attachment objects that represent the attachments in an Outlook item.
Using the Attachments Object

Use the Attachments property to return the Attachments collection for any Outlook item (except notes).

Use the Add method to add an attachment to an item.

To ensure consistent results, always save an item before adding or removing objects in the Attachments collection of the item.

The following Visual Basic for Applications example creates a new mail message, attaches a Q496.xls as an attachment (not a link), and gives the attachment a descriptive caption.

```vba
Set myOlApp = CreateObject("Outlook.Application")
Set myItem = myOlApp.CreateItem(olMailItem)
myItem.Save
Set myAttachments = myItem.Attachments
myAttachments.Add "C:\My Documents\Q496.xls", _
                   olByValue, 1, "4th Quarter 1996 Results Chart"
```

Use Attachments(index), where index is the index number, to return a single Attachment object.
ContactItem Object

Represents a contact in a contacts folder. A contact can represent any person with whom you have any personal or professional contact.
Using the ContactItem Object

Use the **CreateItem** method to create a **ContactItem** object that represents a new contact.

The following Visual Basic for Applications example returns a new contact.

```vba
Set myOlApp = CreateObject("Outlook.Application")
Set myItem = myOlApp.CreateItem(olContactItem)
```

The following VBScript example returns a new contact.

```vbs
Set myItem = Application.CreateItem(olContactItem)
```

Use **Items(index)**, where *index* is the index number of a contact or a value used to match the default property of a contact, to return a single **ContactItem** object from a Contacts folder.

```vbs
```
DistListItem Object

Represents a distribution list in a contacts folder. A distribution list can contain multiple recipients and is used to send messages to everyone in the list.
Using the DistListItem Object

Use the `CreateItem` method to create a **DistListItem** object that represents a new distribution list. The following Microsoft Visual Basic for Applications example creates and displays a new distribution list.

```vbnet
Set myOlApp = CreateObject("Outlook.Application")
Set myItem = myOlApp.CreateItem(olDistributionListItem)
myItem.Display
```

Use `Items(index)`, where `index` is the index number of an item in a contacts folder or a value used to match the default property of an item in the folder, to return a single **DistListItem** object from a contacts folder (that is, a folder whose default item type is **olContactItem**). The following Visual Basic for Applications example sets the current folder as the contacts folder and displays an existing distribution list named Project Team in the folder.

```vbnet
Set myOlApp = CreateObject("Outlook.Application")
Set myNamespace = myOlApp.GetNamespace("MAPI")
Set myFolder = myNamespace.GetDefaultFolder(olFolderContacts)
myFolder.Display
Set myItem = myFolder.Items("Project Team")
myItem.Display
```
Show All
DocumentItem Object

A DocumentItem object is any document other than a Microsoft Outlook item as an item in an Outlook folder. In common usage, this will be an Office document but may be any type of document or executable file.
Exception Object

The Exception object holds information about one instance of an AppointmentItem object which is an exception to a recurring series. Unlike most of the other Microsoft Outlook objects, the Exception object is a read-only object. This means that you cannot create an Exception object but, rather, the object is created when a property of an AppointmentItem is altered. For example, if you change the Start property of one AppointmentItem, you have created an Exception in AppointmentItem.RecurrencePattern.Exceptions.

Note The Exceptions object is on the RecurrencePattern, not the AppointmentItem object itself.
Using the Exception Object

The Exception object can be accessed from the RecurrencePattern object through the Exceptions object.

In Visual Basic for Applications:

Set myOlApp = CreateObject("Outlook.Application")
Set myNameSpace = myOlApp.GetNameSpace("MAPI")
Set myFolder = myNameSpace.GetDefaultFolder(olFolderCalendar)
Set myItems = myFolder.Items
Set myApptItem = myItems("Daily Meeting")
Set myRecurrencePattern = myApptItem.GetRecurrencePattern
Set myException = myRecurrencePattern.Exceptions.Item(1)

In VBScript:

Set myNameSpace = Application.GetNameSpace("MAPI")
Set myFolder = myNameSpace.GetDefaultFolder(9)
Set myItems = myFolder.Items
Set myApptItem = myItems("Daily Meeting")
Set myRecurrencePattern = myApptItem.GetRecurrencePattern
Set myException = myRecurrencePattern.Exceptions.Item(1)
Exceptions Object

The **Exceptions** object contains a group of **Exception** objects. If you have a recurring **AppointmentItem**, the **RecurrencePattern** object defines the recurrence of these appointments. The **Exceptions** object contains the group of **Exception** objects that define the exceptions to that series of appointments.

**Exception** objects are added to the **Exceptions** object whenever a property in the corresponding **AppointmentItem** object is altered.
Using the Exceptions Object

The following example sets a reference to the **Exceptions** object.

Set myExceptions = myRecurrencePattern.Exceptions
Explorer Object

Multiple objects

Represented by the window in which the contents of a folder are displayed.
Using the Explorer Object

- Use the Item method of the Explorers object to return the object representing a specific explorer.

- Use the ActiveExplorer method to return the object representing the currently active explorer (if there is one).

- Use the GetExplorer method to return the Explorer object associated with a folder.

- Use the Display method of a MAPIFolder object to display a folder in its associated explorer.
Explorers Object

The Explorers object contains a set of Explorer objects representing all explorers. An explorer need not be visible to be included in the collection.
Using the Explorers Object

Use the Explorers property to return the Explorers object from the Application object. The following example shows how to retrieve the Explorers object in Microsoft Visual Basic or Microsoft Visual Basic for Applications.

Dim myOlApp as New Outlook.Application
Set myExplorers = myOLApp.Explorers

The following example shows how to retrieve the Explorers object in VBScript.

Set myExplorers = Application.Explorers
Folders Object

Multiple objects

- Folders
- Multiple objects

An object containing MAPIFolder objects that represent all the available Microsoft Outlook folders in a specific subset at one level of the folder tree.
Using the Folders Object

Use the **Folders** property to return the **Folders** object from a **NameSpace** object or another **MAPIFolder** object.

Use **Folders(index)**, where *index* is the name or index number, to return a single **MAPIFolder** object. Folder names are case-sensitive.

The following Visual Basic for Applications example returns the folder named Old Contacts.

```vba
Set myOlApp = CreateObject("Outlook.Application")
Set myNameSpace = myOlApp.GetNameSpace("MAPI")
Set myFolder = _
    myNameSpace.GetDefaultFolder(olFolderContacts)
Set myNewFolder = myFolder.Folders("Old Contacts")
```

The following Visual Basic for Applications example returns the first folder.

```vba
Set myNewFolder = myFolder.Folders(1)
```
FormDescription Object

- **Multiple objects**
- **FormDescription**
- **NameSpace**

Contains the general properties of a Microsoft Outlook **form**. The properties of an Outlook form are displayed on the **Properties** page of a form in design time.

To see the **Properties** page in design time, open the Outlook **item**, select **Forms** on the **Tools** menu, click **Design This Form**, and then click the **Properties** tab in the item's window.
Using the FormDescription Object

Use the `FormDescription` property to return the `FormDescription` object associated with an Outlook item.
Inspector Object

Multiple objects  Inspector

Multiple objects

Represents the window in which an Outlook item is displayed.
Using the Inspector Object

- Use the `ActiveInspector` method to return the object representing the currently active inspector (if there is one).

- Use the `GetInspector` property to return the `Inspector` object associated with an item.

- Use the `Display` method to display an item in its associated inspector.

The Inspector Object and Microsoft Word as the User’s E-mail Editor

If the user’s default e-mail editor is Microsoft Word, and if the message format of the item being edited is plain text or HTML, then no `Inspector` object is associated with the item.
Inspectors Object

The **Inspectors** object contains a set of **Inspector** objects representing all inspectors. An inspector need not be visible to be included in the collection.
Using the Inspectors Object

Use the **Inspectors** property to return the **Inspectors** object from the **Application** object. The following example shows how to retrieve the **Inspectors** object in Microsoft Visual Basic or Microsoft Visual Basic for Applications.

```vba
Dim myOlApp as New Outlook.Application
Set myInspectors = myOlApp.Inspectors
```

The following example shows how to retrieve the **Inspectors** object in VBScript.

```vbscript
Set myInspectors = Application.Inspectors
```
ItemProperties Collection

Multiple objects

A collection of all properties associated with the item.
Using the ItemProperties collection

Use the **ItemProperties** property to return the **ItemProperties** collection. Use `ItemProperties.Item(index)`, where `index` is the name of the object or the numeric position of the item within the collection, to return a single **ItemProperty** object. The following example creates a new **MailItem** object and stores its **ItemProperties** collection in a variable called `objItems`.

```vba
Sub ItemProperty()
    'Creates a new MailItem and access its properties

    Dim olApp As Outlook.Application
    Dim objMail As MailItem
    Dim objItems As ItemProperties
    Dim objItem As ItemProperty

    Set olApp = Outlook.Application
    'Create the mail item
    Set objMail = olApp.CreateItem(olMailItem)
    'Create a reference to the item properties collection
    Set objItems = objMail.ItemProperties
    'Create a reference to the item property page
    Set objItem = objItems.item(0)

End Sub
```

Use the **Add** method to add a new item property to the ItemProperties collection. Use the **Remove** method to remove an item property from the ItemProperties collection.

**Note** You can only add or remove custom properties. Custom properties are denoted by the **IsUserProperty**.

**Note** The **ItemProperties** collection is zero-based, meaning that the first item in the collection is referenced by 0, instead of 1.
ItemProperty Object

Contains information about a given item property. Each item property defines a certain attribute of the item, such as the name, type, or value of the item. The ItemProperty object is a member of the ItemProperties collection.
Using the ItemProperty object

Use ItemProperties.Item(index), where index is the object's numeric position within the collection or it's name to return a single ItemProperty object. The following example creates a reference to the first ItemProperty object in the ItemProperties collection.

Sub NewMail()
'Creates a new MailItem and references the ItemProperties collection

    Dim olApp As Outlook.Application
    Dim objMail As MailItem
    Dim objitems As ItemProperties
    Dim objitem As ItemProperty

    Set olApp = Outlook.Application
    'Create a new mail item
    Set objMail = olApp.CreateItem(olMailItem)
    'Create a reference to the ItemProperties collection
    Set objitems = objMail.ItemProperties
    'Create reference to the first object in the collection
    Set objitem = objitems.item(0)

End Sub
Items Object

- **MAPIFolder**
  - **Items**
  - **NameSpace**

An object containing Outlook item objects in a folder.
Using the Items Object

Use the **Items** property to return the **Items** object of a **MAPIFolder** object.

Use **Items(index)**, where *index* is the name or index number, to return a single Outlook item.

The following Visual Basic for Applications example returns the first item with the subject Need your advice.

```vba
Set myOlApp = CreateObject("Outlook.Application")
Set myNameSpace = myOlApp.GetNameSpace("MAPI")
Set myFolder = _
    myNameSpace.GetDefaultFolder(olFolderInbox)
Set myItem = myFolder.Items("Need your advice")
```

The following Visual Basic for Applications example returns the first item in the folder.

```vba
Set myOlApp = CreateObject("Outlook.Application")
Set myNameSpace = myOlApp.GetNameSpace("MAPI")
Set myFolder = _
    myNameSpace.GetDefaultFolder(olFolderInbox)
Set myItem = myFolder.Items(1)
```

The following VBScript example returns the first item in the folder.

```vbs
Set myNameSpace = Application.GetNameSpace("MAPI")
Set myFolder = myNameSpace.GetDefaultFolder(6)
Set myItem = myFolder.Items(1)
```
JournalItem Object

Multiple objects

- JournalItem
- Multiple objects

Represents a journal entry in a Journal folder. A journal entry represents a record of all Microsoft Outlook-moderated transactions for any given period.
Using the JournalItem Object

Use the `CreateItem` method to create a `JournalItem` object that represents a new journal entry. The following example returns a new journal entry.

```
Set myOlApp = CreateObject("Outlook.Application")
Set myItem = myOlApp.CreateItem(olJournalItem)
```

Use `Items(index)`, where `index` is the index number of a journal entry or a value used to match the default property of a journal entry, to return a single `JournalItem` object from a Journal folder.
Link Object

Represents an item that is linked to another Microsoft Outlook item. Each item has a Links object associated with it that represents all the items that have been linked to the item.

**Note**  For Outlook 2000 and later, only contacts can be linked to other items.
Using the Link Object

Use the **Item** method to retrieve the **Link** object from a **Links** object. Because the **Name** property is the default property of the **Link** object, you can identify the linked item by name.

Set `myLink = myLinks.Item("Microsoft Corporation")`
Links Object

The **Links** object contains a set of **Link** objects representing all **items** linked to a particular Microsoft Outlook item.
Using the Links Object

Use the Links property to return the Links object from the item object; for example:

Set myLinks = myItem.Links
MailItem Object

Multiple objects

Represents a mail message in an Inbox (mail) folder.
Using the MailItem Object

Use the `CreateItem` method to create a `MailItem` object that represents a new mail message. The following example creates and displays a new mail message.

```vba
Set myOlApp = CreateObject("Outlook.Application")
Set myItem = myOlApp.CreateItem(olMailItem)
myItem.Display
```

Use `Items(index)`, where `index` is the index number of a mail message or a value used to match the default property of a message, to return a single `MailItem` object from an Inbox folder. The following example sets the current folder as the Inbox and displays the second mail message in the folder.

```vba
Set myOlApp = CreateObject("Outlook.Application")
Set myNamespace = myOlApp.GetNamespace("MAPI")
Set myFolder = myNamespace.GetDefaultFolder(olFolderInbox)
myFolder.Display
Set myItem = myFolder.Items(2)
myItem.Display
```
MAPIFolder Object

Multiple objects  MAPIFolder  Multiple objects

Represents a Microsoft Outlook folder. A MAPIFolder object can contain other MAPIFolder objects, as well as Outlook items. You can navigate nested folders by using a combination of Folders(index), which returns a folder within a name space or another folder, and the Parent property, which returns the containing object.
Using the MAPIFolder Object

Use **Folders**(*index*), where *index* is the name or index number, to return a single MAPIFolder object from a NameSpace object or another MAPIFolder object.

There is a set of folders within an Outlook data store that support the default functionality of Outlook. Use **GetDefaultFolder**(*index*), where *index* is one of the OlDefaultFolders constants to return one of the default Outlook folders in the Outlook NameSpace object. The OlDefaultFolders constants are olFolderCalendar, olFolderContacts, olFolderDeletedItems, olFolderDrafts, olFolderInbox, olFolderJournal, olFolderNotes, olFolderOutbox, olFolderSentMail, and olFolderTasks.

Use the **Add** method to add a folder to the Folders object. The Add method has an optional argument that can be used to specify the type of items that can be stored in that folder. By default, folders created inside another folder inherit the type of the parent folder.

Folders within the Outlook data store can be typed; for example, the Calendar folder will only contain AppointmentItem objects and the Contacts folder will only contain ContactItem and DistListIItem objects.

Note that when items of a specific type are saved, they are saved directly into their corresponding default folder. For example, when the **GetAssociatedAppointment** method is applied to a MeetingItem in the Inbox folder, the appointment that is returned will be saved to the default Calendar folder.
MeetingItem Object

Multiple objects

Represents an item in an Inbox (mail) folder. A MeetingItem object represents a change to the recipient's Calendar folder initiated by another party or as a result of a group action.
Using the MeetingItem Object

Unlike other Microsoft Outlook objects, you cannot create this object. It is created automatically when you set the `MeetingStatus` property of an `AppointmentItem` object to `olMeeting` and send it to one or more users. They receive it in their inboxes as a `MeetingItem`.

The following example uses the `CreateItem` method to create an appointment. It becomes a `MeetingItem` with both a required and an optional attendee when it is received in the inbox of each of the recipients.

```
Set myItem = myOlApp.CreateItem(olAppointmentItem)
myItem.MeetingStatus = olMeeting
myItem.Subject = "Strategy Meeting"
myItem.Location = "Conference Room B"
myItem.Start = #9/24/97 1:30:00 PM#
myItem.Duration = 90
Set myRequiredAttendee = myItem.Recipients.Add("Nate _ Sun")
myRequiredAttendee.Type = olRequired
Set myOptionalAttendee = myItem.Recipients.Add("Kevin _ Kennedy")
myOptionalAttendee.Type = olOptional
Set myResourceAttendee = _
    myItem.Recipients.Add("Conference Room B")
myResourceAttendee.Type = olResource
myItem.Send
```

Use the `GetAssociatedAppointment` method to return the `AppointmentItem` object associated with a `MeetingItem` object, and work directly with the `AppointmentItem` object to respond to the request.
**NameSpace Object**

Multiple objects  
  \[\text{NameSpace}\]  
  \[\text{Multiple objects}\]

Represents an abstract root object for any data source. The object itself provides methods for logging in and out, accessing storage objects directly by ID, accessing certain special default folders directly, and accessing data sources owned by other users.
Using the NameSpace Object

Use `GetNamespace("MAPI")` to return the Outlook NameSpace object from the Application object.

The only data source supported is MAPI, which allows access to all Outlook data stored in the user's mail stores.
NoteItem Object

Multiple objects

Represents a note in a Notes folder.

A NoteItem is not customizable. If you open a new note, you will notice that it is not possible to place it in design time.

The Subject property of a NoteItem object is read-only because it is calculated from the body text of the note. Also, the NoteItem Body can only be rich text, so the properties that correspond to HTML and Microsoft Word content do not apply. Although the GetInspector property will work on notes, because notes can't be customized, some of the Inspector properties and methods will not apply to a NoteItem.
Using the NoteItem Object

Use the `CreateItem` method to create a `NoteItem` object that represents a new note. The following Microsoft Visual Basic example returns a new note.

```vba
Set myOlApp = CreateObject("Outlook.Application")
Set myItem = myOlApp.CreateItem(olNoteItem)
```

The following example shows how to create a `NoteItem` object using VBScript.

```vbs
Set myItem = Application.CreateItem(5)
```

Use `Items(index)`, where `index` is the index number of a note or a value used to match the default property of a note, to return a single `NoteItem` object from a Notes folder.
OutlookBarGroup Object

Multiple objects

- OutlookBarGroup
- Multiple objects

Represents a group of shortcuts in the Outlook Bar of an explorer window.
Using the OutlookBarGroup Object

Use the **Item** method to retrieve the **OutlookBarGroup** object from an **OutlookBarGroups** object. Because the **Name** property is the default property of the **OutlookBarGroup** object, you can identify the group by name. For example:

```vba
Set myOlBarGroup = myOutlookBarGroups.Item("Other Shortcuts")
```
OutlookBarGroups Object

OutlookBarStorage ← OutlookBarGroups

Multiple objects

The OutlookBarGroups object contains a set of OutlookBarGroup objects representing all groups in the Outlook Bar.
Using the OutlookBarGroups Object

Use the Groups property to return the OutlookBarGroups object from the OutlookBarStorage object. For example:

Set myGroups = myOutlookBarStorage.Groups
OutlookBarPane Object

OutlookBarPane

Multiple objects

Represents the Microsoft Outlook pane in an explorer window. The Outlook Bar contains shortcuts to Microsoft Outlook and file-system folders arranged in groups.
Using the OutlookBarPane Object

Use the Item method to retrieve the OutlookBarPane object from a Panes object. Because the Name property is the default property of the OutlookBarPane object, you can identify the Outlook Bar pane by name. For example:

Set myOlBarPane = myPanes.Item("OutlookBar")
OutlookBarShortcut Object

OutlookBarShortcuts \- OutlookBarShortcut
\- NameSpace

Represents a shortcut in a group in the Outlook Bar of an explorer window.
Using the OutlookBarShortcut Object

Use the Item method to retrieve the OutlookBarShortcut object from an OutlookBarShortcuts object. Because the Name property is the default property of the OutlookBarShortcut object, you can identify the shortcut by name. For example:

Set myOlBarShortcut = myOutlookBarShortcuts.Item("Calendar")
OutlookBarShortcuts Object

The OutlookBarShortcuts collection contains a set of OutlookBarShortcut objects representing all shortcuts in an Outlook Bar group.
Using the OutlookBarShortcuts Object

Use the Shortcuts property to return the OutlookBarShortcuts collection object from the OutlookBarGroup object. Foreexample:

```
Set myShortcuts = myOutlookBarGroup.Shortcuts
```
OutlookBarStorage Object

OutlookBarPane ▼ OutlookBarStorage
▼ Multiple objects

Represents the storage for objects in the Microsoft Outlook Bar pane.
Using the OutlookBarStorage Object

Use the **Contents** property of an **OutlookBarPane** object to retrieve the **OutlookBarStorage** object for the pane. For example:

```vba
Set myOLBarStorage = myPanes.Item("OutlookBar").Contents
```

Use the **Groups** property to retrieve the **OutlookBarGroups** object for the Outlook Bar.
Pages Object

Pages ⊲ NameSpace

An object containing pages that represent the pages of an Inspector window. Every Inspector object has a Pages object defined, which is empty (count 0) if the Outlook item has never been customized before.
Using the Pages Object

Use the **ModifiedFormPages** property to return the **Pages** object from an **Inspector** object. The following example returns the **Pages** object for the active Inspector.

```vbscript
Set myPages = myItem.GetInspector.ModifiedFormPages
```

Use the **Add** method to create a custom page (you can add as many as 5 customizable pages). Use the **Name** argument of the **Add** method to set the display name of the returned page. In addition to adding custom pages, you can use the **Name** argument to return the main page of an **Inspector** object for modification.

The following example returns a custom page with a default name (such as "Custom1").

```vbscript
Set myPage = myPages.Add
```

The following example returns a custom page named "My Page."

```vbscript
Set myPage = myPages.Add("My Page")
```

The following example returns the Message page if the Inspector contains a mail message.

```vbscript
Set myPage = myPages.Add("Message")
```

The following example returns the General (main) page if the inspector contains a contact.

```vbscript
Set myPage = myPages.Add("General")
```

Use **ModifiedFormPages(index)**, where **index** is the name or index number, to return a single page from a **Pages** object.
The Panes object contains the panes displayed by the specified Explorer.
Using the Panes Object

Use the **Panes** property to return the **Panes** collection object from an **Explorer** object.

Set `myPanes = myExplorer.Panes`

Use the **Item** method to retrieve a specific pane. To retrieve the **OutlookBarPane** object representing the Outlook Bar, use the following:

Set `myOLBarPane = myExplorer.Panes.Item("OutlookBar")`

For Microsoft Outlook 2000 and later, the Outlook Bar is the only pane that you can access through the **Panes** object.
PostItem Object

Application – PostItem
   – Multiple objects

Represents a post in a public folder that others may browse. Unlike a MailItem object, a PostItem object is not sent to a recipient. You use the Post method, which is analogous to the Send method for the MailItem object, to save the PostItem to the target public folder instead of mailing it.
Using the PostItem Object

Use the `CreateItem` or `CreateItemFromTemplate` method to create a PostItem object that represents a new post. The following example returns a new post.

Set `myItem = myOlApp.CreateItem(olPostItem)`

Use `Items(index)`, where `index` is the index number of a post or a value used to match the default property of a post, to return a single PostItem object from a public folder.
PropertyPage Object

PropertyPages — PropertyPage

Represents a custom property page in the Microsoft Outlook Options dialog box or in the folder Properties dialog box. Outlook uses this object to allow a custom property page to interact with the Apply button in the dialog box.
Using the PropertyPage Object

The **PropertyPage** object is an abstract object. That is, the **PropertyPage** object in the Microsoft Outlook Object Library contains no implementation code. Instead, it is provided as a template to help you implement the object in Microsoft Visual Basic. This provides a predefined set of interfaces that Outlook can use to determine whether your custom property page has changed and to notify your program that the user has clicked the **Apply** or **OK** button. (If your custom property page does not rely on the **Apply** button, then you do not need to implement the **PropertyPage** object.)

A custom property page is an ActiveX control that is displayed by Outlook in the **Options** dialog box or in the folder **Properties** dialog box when the user clicks on the custom property page’s tab. To implement the **PropertyPage** object, the module that contains the implementation code must contain the following statement:

```vbnet
Implements Outlook.PropertyPage
```

The module must also contain procedures that implement the properties and methods of the **PropertyPage** object. For example, to implement the **Dirty** property, a procedure similar to the following appears in the module.

```vbnet
Private Property Get PropertyPage_Dirty() As Boolean
    PropertyPage_Dirty = gblDirty
End Property
```

To implement a method of the **PropertyPage** object, the module must contain a statement similar to the following.

```vbnet
Private Sub PropertyPage_Apply()
    ' Code to set properties according to the user's selections goes here.
End Sub
```
PropertyPages Object

The PropertyPages object contains the custom property pages that have been added to the Microsoft Outlook Options dialog box or to the folder Properties dialog box.
**Using the PropertyPages Object**

You receive a `PropertyPages` object as a parameter of the `OptionsPagesAdd` event. Use the `Add` method to add a `PropertyPage` object to the `PropertyPages` object.

**Note** If more than one program handles the `OptionsPagesAdd` event, the order in which the programs receive the event (and therefore, the order in which pages are added to the `PropertyPages` object) cannot be guaranteed.
PropertyPageSite Object

PropertyPageSite \textarrow{\rightarrow} NameSpace

Represents the container of a custom property page.
Using the PropertyPageSite Object

Use the `Parent` property of the ActiveX control that implements the `PropertyPage` object associated with the `PropertyPageSite` object to return the `PropertyPageSite` object. The Declarations section of the module implementing the `PropertyPage` object must contain a declaration similar to the following.

```vbnet
Private myPropertyPageSite As Outlook.PropertyPageSite
```

The object is then returned from the `Parent` property.

```vbnet
Set myPropertyPageSite = Parent
```

Use the `OnStatusChange` method to notify Microsoft Outlook that the property page has changed.
Recipient Object

Multiple objects

Represents a user or resource in Outlook, generally a mail message addressee.
Using the Recipient Object

Use **Recipients**(*index*), where *index* is the name or index number, to return a single **Recipient** object.

Use the **Add** method to create a new **Recipient** object and add it to the **Recipients** object. The **Type** property of a new **Recipient** object is set to the default for the associated **AppointmentItem**, **JournalItem**, **MailItem**, **MeetingItem** or **TaskItem** object and must be reset to indicate another recipient type.

The following Visual Basic for Applications example creates a new **MailItem** object and adds Jon Grande as the recipient using the default type ("To").

```
Set myOlApp = CreateObject("Outlook.Application")
Set myItem = myOlApp.CreateItem(olMailItem)
Set myRecipient = myItem.Recipients.Add ("Jon Grande")
```

The following Visual Basic for Applications example creates the same **MailItem** object as the preceding example, and then changes the type of the **Recipient** object from the default (To) to CC.

```
Set myOlApp = CreateObject("Outlook.Application")
Set myItem = myOlApp.CreateItem(olMailItem)
Set myRecipient = myItem.Recipients.Add ("Jon Grande")
myRecipient.Type = olCC
```
Recipients Object

Multiple objects \$\text{Recipients}\$

\$\text{Multiple objects}\$

The \textbf{Recipients} object contains \textbf{Recipient} objects.
Using the Recipients Object

Use the **Recipients** property to return the **Recipients** object of an **AppointmentItem**, **JournalItem**, **MailItem**, **MeetingItem** or **TaskItem** object.

Use the **Add** method to create a new **Recipient** object and add it to the **Recipients** object. The **Type** property of a new **Recipient** object is set to the default for the associated **AppointmentItem**, **JournalItem**, **MailItem** or **TaskItem** object and must be reset to indicate another recipient type.

The following example creates a new **MailItem** object and adds Jon Grande as the recipient using the default type ("To").

```vba
Set myOlApp = CreateObject("Outlook.Application")
Set myItem = myOlApp.CreateItem(olMailItem)
Set myRecipient = myItem.Recipients.Add ("Jon Grande")
```

The following example creates the same **MailItem** object as the preceding example, and then changes the type of the **Recipient** object from the default ("To") to CC.

```vba
Set myOlApp = CreateObject("Outlook.Application")
Set myItem = myOlApp.CreateItem(olMailItem)
Set myRecipient = myItem.Recipients.Add ("Jon Grande")
myRecipient.Type = olCC
```

Use **Recipients(index)**, where **index** is the name or index number, to return a single **Recipient** object.
RecurrencePattern Object

Multiple objects

Represents the pattern of incidence of recurring appointments and tasks for the associated AppointmentItem and TaskItem object.
Using the RecurrencePattern Object

Use the `GetRecurrencePattern` method to return the `RecurrencePattern` object associated with an `AppointmentItem` or `TaskItem` object.

Calling `GetRecurrencePattern` or `ClearRecurrencePattern` has the side effect of setting the `IsRecurring` property of the item accordingly. This property can be used as required for efficient filtering of the `Items` object.

The type of recurrence pattern is indicated by the `RecurrenceType` property. The `RecurrenceType` property is the first property you should set.

The following properties are valid for all recurrence patterns: `EndTime`, `Occurrences`, `StartDate`, `StartTime`, or `Type`.

The following table shows the properties that are valid for the different recurrence types. The properties listed are not all required for the given type; an error occurs if the item is saved and the property is null or contains an invalid value. Monthly and yearly patterns are only valid for a single day. Weekly patterns are only valid as the `Or` of the `DayOfWeekMask`.

<table>
<thead>
<tr>
<th>RecurrenceType</th>
<th>Properties</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>olRecursDaily</td>
<td>Interval</td>
<td>Every N days</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Every Tuesday, Wednesday, and Thursday</td>
</tr>
<tr>
<td></td>
<td>DayOfWeekMask</td>
<td></td>
</tr>
<tr>
<td>olRecursMonthly</td>
<td>Interval</td>
<td>Every N months</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The Nth day of the month</td>
</tr>
<tr>
<td></td>
<td>DayOfMonth</td>
<td></td>
</tr>
<tr>
<td>olRecursMonthNth</td>
<td>Interval</td>
<td>Every N months</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The Nth Tuesday</td>
</tr>
<tr>
<td></td>
<td>Instance</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>DayOfWeekMask</td>
<td>Every Tuesday and Wednesday</td>
</tr>
<tr>
<td>olRecursWeekly</td>
<td>Interval</td>
<td>Every N weeks</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Every Tuesday, Wednesday, and Thursday</td>
</tr>
<tr>
<td></td>
<td>DayOfWeekMask</td>
<td></td>
</tr>
<tr>
<td>olRecursYearly</td>
<td>DayOfMonth</td>
<td>Thursday</td>
</tr>
<tr>
<td>---------------</td>
<td>---------------------</td>
<td>-------------------------------</td>
</tr>
<tr>
<td></td>
<td>MonthOfYear</td>
<td>The Nth day of the month</td>
</tr>
<tr>
<td></td>
<td>Instance</td>
<td>February</td>
</tr>
<tr>
<td>olRecursYearNth</td>
<td>DayOfWeekMask</td>
<td>The Nth Tuesday</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Tuesday, Wednesday, Thursday</td>
</tr>
<tr>
<td></td>
<td>MonthOfYear</td>
<td>February</td>
</tr>
</tbody>
</table>
**Reminder Object**

Reminders → Reminder → NameSpace

 Represents a Microsoft Outlook reminder. Reminders allow users to keep track of upcoming appointments by scheduling a pop-up dialog box to appear at a given time. In addition to appointments, reminders can occur for tasks, contacts and e-mail messages.
Using the Reminder object

Use **Reminders**(index), where index is the name or index number of the reminder, to return a single **Reminder** object. The following example displays the caption of the first reminder in the collection.

Sub ViewReminderInfo()
'Displays information about first reminder in collection

    Dim olApp As Outlook.Application
    Dim objRem As Reminder

    Set olApp = Outlook.Application
    'If there are reminders, display message
    If olApp.Reminders.Count <> 0 Then
        Set objRem = olApp.Reminders.Item(1)
        MsgBox "The caption of the first reminder in the collection is:
        objRem.Caption"
    Else
        MsgBox "There are no reminders in the collection."
    End If

End Sub

Reminders are created programmatically when a new Microsoft Outlook item, such as an **AppointmentItem** object, is created and the item's **ReminderSet** property is set to **True**. Use the item's **ReminderTime** property to set the time in minutes at which the reminder will occur. The following example creates a new meeting and sets the **ReminderSet** property to **True**, adding a new **Reminder** object to the **Reminders** collection.

Sub AddMeeting()
'Adds a new meeting and reminder to the reminders collection

    Dim olApp As Outlook.Application
    Dim objMeet As AppointmentItem

    Set olApp = Outlook.Application
    Set objMeet = olApp.CreateItem(olAppointmentItem)

    objMeet.ReminderSet = True
    objMeet.Subject = "Tuesday's meeting"
Use the **Reminders** collection's **Remove** method to remove a **Reminder** object from the collection. Once a reminder is removed from its associated item, the **AppointmentItem** object's **ReminderSet** property is set to **False**.
Reminders Collection

A collection of all the Reminder objects in a Microsoft Outlook application that represents the reminders for all pending appointment items.
Using the Reminders collection

Use the **Application** object's **Reminders** property to return the **Reminders** collection. Use **Reminders**(index), where *index* is the name or ordinal value of the reminder, to return a single **Reminder** object. The following example displays the captions of each reminder in the list.

```vba
Sub ViewReminderInfo()
    'Lists reminder caption information
    Dim olApp As Outlook.Application
    Dim objRem As Reminder
    Dim objRems As Reminders
    Dim strTitle As String
    Dim strReport As String

    Set olApp = Outlook.Application
    Set objRems = olApp.Reminders
    strTitle = "Current Reminders:
    'If there are reminders, display message
    If olApp.Reminders.Count <> 0 Then
        For Each objRem In objRems
            'If string is empty, create new string
            If strReport = "" Then
                strReport = objRem.Caption & vbCrLf
            Else
                'Add info to string
                strReport = strReport & objRem.Caption & vbCrLf
            End If
        Next objRem
        'Display report in dialog
        MsgBox strTitle & vbCrLf & vbCrLf & strReport
    Else
        MsgBox "There are no reminders in the collection."
    End If

End Sub
```

Reminders are created programmatically when a new Microsoft Outlook item is created with a reminder. For example, a reminder is created when an **AppointmentItem** object is created and the **AppointmentItem** object's **ReminderSet** property is set to **True**. Use the **AppointmentItem** object's
**ReminderTime** property to set the time in minutes at which the reminder will occur. The following example creates a new meeting and sets the ReminderSet property to **True**, adding a new **Reminder** object to the **Reminders** collection.

Sub AddMeeting()
' Adds a new meeting and reminder to the reminders collection

    Dim olApp As Outlook.Application
    Dim objMeet As AppointmentItem

    Set olApp = Outlook.Application
    Set objMeet = olApp.CreateItem(olAppointmentItem)

    objMeet.ReminderSet = True
    objMeet.Subject = "Tuesday's meeting"

End Sub
Show All
RemoteItem Object

RemoteItem

Multiple objects

Represents a remote item in an Inbox (mail) folder. The RemoteItem object is similar to the MailItem object, but it contains only the Subject, Received Date and Time, Sender, Size and the first 256 characters of the body of the message. It is used to give someone connecting in remote mode enough information to decide whether or not to download the corresponding mail message.
Using the RemoteItem Object

Unlike other Microsoft Outlook objects, you cannot create this object. Remote items are created by Outlook automatically when you use a Remote Access System (RAS) connection. Each RemoteItem object created on the local system corresponds to a pre-existing MailItem object on the remote system.

The RemoteItem object inherits a number of properties, methods, and events that, because of the nature of the object, have no function. The Object Browser shows these properties, methods, and events as belonging to the RemoteItem object, but attempting to use them will produce no effect.

The methods that do not work for the RemoteItem object include Close, Copy, Display, Move, and Save.

The properties that do not work for the RemoteItem object include BillingInformation, Body, Categories, Companies, and Mileage.

The events that do not work for the RemoteItem object include Open, Close, Forward, Reply, ReplyAll, and Send.
ReportItem Object

Application ReportItem
  - Multiple objects

Represents a mail-delivery report in an Inbox (mail) folder. The ReportItem object is similar to a MailItem object, and it contains a report (usually the non-delivery report) or error message from the mail transport system.
Using the ReportItem Object

Unlike other Microsoft Outlook objects, you cannot create this object. Report items are created automatically when any report or error in general is received from the mail transport system.
Results Collection

Stores data and results returned by the Search object and the AdvancedSearch method. The Results object contains properties and methods that allow you to view and manipulate data. For example the GetNext, GetPrevious, GetFirst, and GetLast methods allow you to search through the results and view the data by field. The Sort method allows you to sort the data.
Using the Results Collection

Use the `SearchObject.Results` property to return a `Results` object. The following event procedure stores the results of a search in a variable named `objRsts` and displays the results of the search in the Immediate window.

```vba
Private Sub Application_AdvancedSearchComplete(ByVal SearchObject As Search)
    Dim objRsts As Results
    MsgBox "The search " & SearchObject.Tag & 
        "has completed. The scope of the search was " & 
            SearchObject.Scope & "."
    Set objRsts = SearchObject.Results
    ' Print out number in Results collection
    Debug.Print objRsts.Count
    ' Print out each member of Results collection
    For Each Item In objRsts
        Debug.Print Item
    Next
End Sub
```
Search Object

Contains information about individual searches performed against Microsoft Outlook items. The **Search** object contains properties that define the type of search and the parameters of the search itself.
Using the Search object

Use the Application object's AdvancedSearch method to return a Search object. The following example returns a search object named "SubjectSearch" and displays the object's Tag and Filter property values. The Tag property is used to identify a specific search once it has completed.

Sub SearchInboxFolder()
'Searches the Inbox

    Dim objSch As Search
    Const strF As String = _
        "urn:schemas:mailheader:subject = 'Office Christmas Party'"
    Const strS As String = "Inbox"
    Const strTag As String = "SubjectSearch"
    Set objSch = Application.AdvancedSearch(Scope:=strS, _
        Filter:=strF, SearchSubFolders:=True, Tag:=strTag)

End Sub

Use the AdvancedSearchComplete event to determine when a given search has completed. The following example displays information about the search and the results of the search.

Private Sub Application_AdvancedSearchComplete(ByVal SearchObject As

    Dim objRsts As Results
    MsgBox "The search " & SearchObject.Tag & " has completed.
    Set objRsts = SearchObject.Results
    'Print out number in Results collection
    Debug.Print objRsts.Count
    'Print out each member of Results collection
    For Each Item In objRsts
        Debug.Print Item
    Next

End Sub
Selection Object

The **Selection** object contains a set of Microsoft Outlook items representing the **items** currently selected in an explorer.
Using the Selection Object

Use the Selection property to return the Selection collection from the Explorer object. For example:

Set mySelectedItems = myExplorer.Selection
SyncObject Object

SyncObjects | SyncObject
--- | ---
NameSpace

Represents a synchronization profile for a user. A synchronization profile lets users configure different synchronization scenarios, selecting which folders and which filters apply.
Using the SyncObject Object

Use the Item method to retrieve the SyncObject object from a SyncObjects object. Because the Name property is the default property of the SyncObject object, you can identify the group by name. For example:

Set mySyncObject = mySyncObjects.Item("Daily")

The SyncObject object is read-only; you cannot change its properties or create new ones.
SyncObjects Object

The SyncObjects object contains a set of SyncObject objects representing the synchronization profiles for a user.
Using the SyncObjects Object

Use the SyncObjects property to return the SyncObjects object from a NameSpace object. For example:

Set mySyncObjects = Application.GetNameSpace("MAPI").SyncObjects

The SyncObjects object is read-only. You cannot add an item to the collection.
TaskItem Object

Multiple objects $\rightarrow$ TaskItem

$\rightarrow$ Multiple objects

Represents a task (an assigned, delegated, or self-imposed task to be performed within a specified time frame) in a Tasks folder.
Using The TaskItem Object

Use the CreateItem method to create a TaskItem object that represents a new task.

The following Visual Basic for Applications example returns a new task.

Set myOlApp = CreateObject("Outlook.Application")
Set myItem = myOlApp.CreateItem(olTaskItem)

The following sample shows how to create a task using VBScript.

Set myItem = Application.CreateItem(3)

Use Items(index), where index is the index number of a task or a value used to match the default property of a task, to return a single TaskItem object from a Tasks folder.
TaskRequestAcceptItem Object

Multiple objects

TaskRequestAcceptItem

Represents an item in an Inbox (mail) folder.

A TaskRequestAcceptItem object represents a response to a TaskRequestItem sent by the initiating user. If the delegated user accepts the task, the ResponseState property is set to olTaskAccept. The associated TaskItem is received by the delegator as a TaskRequestAcceptItem object.
Using the TaskRequestAcceptItem Object

Unlike other Microsoft Outlook objects, you cannot create this object.

Use the `GetAssociatedTask` method to return the `TaskItem` object that is associated with this `TaskRequestAcceptItem`. Work directly with the `TaskItem` object.
TaskRequestDeclineItem Object

TaskRequestDeclineItem

Multiple objects

Represents an item in an Inbox (mail) folder.

A TaskRequestDeclineItem object represents a response to a TaskRequestItem sent by the initiating user. If the delegated user declines the task, the ResponseState property is set to olTaskDecline. The associated TaskItem is received by the delegator as a TaskRequestDeclineItem object.
Using the TaskRequestDeclineItem Object

Unlike other Microsoft Outlook objects, you cannot create this object.

Use the GetAssociatedTask method to return the TaskItem object that is associated with this TaskRequestDeclineItem. Work directly with the TaskItem object.
TaskRequestItem Object

TaskRequestItem

Multiple objects

Represents an item in an Inbox (mail) folder. A TaskRequestItem object represents a change to the recipient's Tasks list initiated by another party or as a result of a group tasking.
Using the TaskRequestItem Object

Unlike other Microsoft Outlook objects, you cannot create this object. When the sender applies the Assign and Send methods to a TaskItem object to assign (delegate) the associated task to another user, the TaskRequestItem object is created when the item is received in the recipient's Inbox.

The following Visual Basic for Applications example creates a simple task, assigns it to another user, and sends it. When the task request arrives in the recipient's Inbox, it is received as a TaskRequestItem.

```vba
Set myOlApp = CreateObject("Outlook.Application")
Set myItem = myOlApp.CreateItem(olTaskItem)
myItem.Assign
Set myDelegate = myItem.Recipients.Add("Jeff Smith")
myItem.Subject = "Prepare Agenda For Meeting"
myItem.DueDate = #9/20/97#
myItem.Send
```

The following example shows how to perform the same task using VBScript.

```vbs
Set myItem = Application.CreateItem(3)
myItem.Assign
Set myDelegate = myItem.Recipients.Add("Jeff Smith")
myItem.Subject = "Prepare Agenda For Meeting"
myItem.DueDate = #9/20/97#
myItem.Send
```

Use the GetAssociatedTask method to return the TaskItem object, and work directly with the TaskItem object to respond to the request.
**TaskRequestUpdateItem Object**

A **TaskRequestUpdateItem** object represents a response to a **TaskRequestItem** sent by the initiating user. If the delegated user updates the task by changing properties such as the **DueDate** or the **Status**, and then sends it, the associated **TaskItem** is received by the delegator as a **TaskRequestUpdateItem** object.
Using the TaskRequestUpdateItem Object

Unlike other Microsoft Outlook objects, you cannot create this object.

Use the GetAssociatedTask method to return the TaskItem object that is associated with this TaskRequestUpdateItem. Work directly with the TaskItem object.
UserProperties Object

Multiple objects

A object containing UserProperty objects that represent the custom properties of an Outlook item.
Using The UserProperties Object

Use the `UserProperties` property to return the `UserProperties` object for an Outlook item.

Use the `Add` method to create a new `UserProperty` for an item and add it to the `UserProperties` object. The `Add` method allows you to specify a name and type for the new property. The following example adds a custom text property named MyPropName to `myItem`.

```vba
Set myProp = myItem.UserProperties.Add("MyPropName", olText)
```

Use `UserProperties(index)`, where `index` is a name or index number, to return a single `UserProperty` object.

When you create a custom property, a field is added in the folder that contains the item (using the same name as the property). That field can be used as a column in folder views.
Show All
UserProperty Object

UserProperties UserProperty NameSpace

Represents a custom property of a Microsoft Outlook item.
Using The UserProperty Object

Use `UserProperties(index)`, where `index` is a name or index number, to return a single `UserProperty` object.

Use the `Add` method to create a new `UserProperty` for an item and add it to the `UserProperties` object. The `Add` method allows you to specify a name and type for the new property. The following example adds a custom text property named `MyPropName`.

Set `myProp = myItem.UserProperties.Add("MyPropName", olText)`

When you create a custom property, a field is added in the folder that contains the item (using the same name as the property). That field can be used as a column in folder views.
View Object

Multiple objects View

NameSpace

The View object allows you to create customizable views that allow you to better sort, group and ultimately view data of all different types. There are a variety of different view types that provide the flexibility needed to create and maintain your important data.

- The table view type (olTableView) allows you to view data in a simple field-based table.

- The Calendar view type (olCalendarView) allows you to view data in a calendar format.

- The card view type (olCardView) allows you to view data in a series of cards. Each card displays the information contained by the item and can be sorted.

- The icon view type (olIconView) allows you to view data as icons, similar to a Windows folder or explorer.

- The timeline view type (olTimelineView) allows you to view data as it is received in a customizable linear time line.

Views are defined and customized using the View object's XML property. The XML property allows you to create and set a customized XML schema that defines the various features of a view.
Using the View object

Use **Views(index)**, where *index* is the name of the View object or its ordinal value, to return a single View object. The following example returns a view called Table View and stores it in a variable of type View called objView.

```vba
Sub GetView()
    'Creates a new view
    Dim olApp As Outlook.Application
    Dim objName As NameSpace
    Dim objViews As Views
    Dim objView As View

    Set olApp = Outlook.Application
    Set objName = olApp.GetNamespace("MAPI")
    Set objViews = objName.GetDefaultFolder(olFolderInbox).Views
    'Return a view called Table View
    Set objView = objViews.Item("Table View")
End Sub
```

Use the **Add** method of the **Views** collection to create a new view. The following example creates a new view of type **olTableView** called New Table.

```vba
Sub CreateView()
    'Creates a new view
    Dim olApp As Outlook.Application
    Dim objName As NameSpace
    Dim objViews As Views
    Dim objNewView As View

    Set olApp = Outlook.Application
    Set objName = olApp.GetNamespace("MAPI")
    Set objViews = objName.GetDefaultFolder(olFolderInbox).Views
    Set objNewView = objViews.Add(Name:="New Table", _
                                   ViewType:=olTableView, SaveOption:=olViewSaveOp
End Sub
```
Views Collection

Multiple objects – Views

- Multiple objects

A collection of all View objects in the current folder.
Using the Views collection

Use the Views property of the Explorer or MAPIFolder objects to return the Views collection. Use Views.Item(index), where index is the object's name or position within the collection, to return a single View object. The following example returns a View object of type olTableView called Table View.

Sub GetView()
'Returns a view called Table View

    Dim olApp As Outlook.Application
    Dim objName As NameSpace
    Dim objViews As Views
    Dim objView As View

    Set olApp = Outlook.Application
    Set objName = olApp.GetNamespace("MAPI")
    Set objViews = objName.GetDefaultFolder(olFolderInbox).Views
    'Return a view called Table View
    Set objView = objViews.Item("Table View")

End Sub

Use the Add method of the views collection to add a new view to the collection. The following example adds a new view of type olIconView in the user's Notes folder.

Note The Add method will fail if a view with the same name already exists.

Sub CreateView()
'Creates a new view

    Dim olApp As Outlook.Application
    Dim objName As NameSpace
    Dim objViews As Views
    Dim objNewView As View

    Set olApp = Outlook.Application
    Set objName = olApp.GetNamespace("MAPI")
    Set objViews = objName.GetDefaultFolder(olFolderNotes).Views
    Set objNewView = objViews.Add(Name:="New Icon View Type", _
        viewType:=olIconView, SaveOption:=olViewSaveOpt
Use the `Remove` method to remove a view from the collection. The following example removes the above view, "New Icon View Type", from the collection.

```vbscript
Sub DeleteView()
' Deletes a view from the collection

    Dim olApp As Outlook.Application
    Dim objName As NameSpace
    Dim objViews As Views
    Dim objNewView As View

    Set olApp = Outlook.Application
    Set objName = olApp.GetNamespace("MAPI")
    Set objViews = objName.GetDefaultFolder(olFolderNotes).Views
    objViews.Remove("New Icon View Type")

End Sub
```
Activate Method

Activates an explorer or inspector window by bringing it to the foreground and setting keyboard focus.

expression.Activate

expression  Required. An expression that returns an Explorer or Inspector object.
Example

This Microsoft Visual Basic/Visual Basic for Applications example responds to the **NewMail** event by activating the explorer window. The sample code must be placed in a class module, and the Initialize_handlers routine must be called before the event procedure can be called by Microsoft Outlook.

```vbnet
Public WithEvents myOlApp As Outlook.Application
Public WithEvents myOlExp As OutlookExplorer

Public Sub Initialize_handlers()
    Set myOlApp = CreateObject("Outlook.Application")
    Set myOlExp = myOlApp.ActiveExplorer
End Sub

Private Sub myOlApp_NewMail()
    myOlExp.Activate
End Sub
```
ActiveExplorer Method

Returns the topmost Explorer object on the desktop. If no explorer is active, returns Nothing. Use this method to return the Explorer object that the user is most likely viewing. This method is also useful for determining when there is no active explorer, so a new one can be opened.

expression.ActiveExplorer

type expression  Required. An expression that returns an Application object.
Example

This Visual Basic for Applications example uses **ActiveExplorer** to demonstrate how to obtain an **Explorer** object for use, preferably getting the user's active explorer, but creating a new one for `myFolder` if there isn't already one on the desktop.

```vba
Set myOlApp = CreateObject("Outlook.Application")
Set myExplorer = myOlApp.ActiveExplorer
If TypeName(myExplorer) = "Nothing" Then
    Set myExplorer = myFolder.GetExplorer
End If
```
ActiveInspector Method

Returns the topmost Inspector object on the desktop. If no inspector is active, returns Nothing. Use this method to access the Inspector object that the user is most likely to be viewing.

expression.ActiveInspector

expression   Required. An expression that returns an Application object
Remarks

If the user’s default e-mail editor is Microsoft Word, and if the message format of the item being edited is plain text or HTML, the ActiveInspector method produces an error and returns Nothing.
Example

This Visual Basic for Applications example uses the **ActiveInspector** method to demonstrate how to obtain the currently active **Inspector** object and display the name of the item that the inspector is displaying.

```vba
Sub GetInspector()
    'Displays the subject of the active Inspector
    Dim myolapp As Outlook.Application
    Dim myinspector As Inspector
    Set myolapp = CreateObject("Outlook.Application")
    Set myinspector = myolapp.ActiveInspector

    'Test if an inspector is active
    If Not TypeName(myinspector) = "Nothing" Then
        'Display subject of active inspector
        MsgBox "The active item is " & myinspector.CurrentItem.Subject
    End If
End Sub

If you use VBScript, you do not create the **Application** object, and you cannot use named constants. This example shows how to perform the same task using VBScript.

```vba
Sub ActiveInspector()
    'Displays the name of the active Inspector
    Set myinspector = Application.ActiveInspector
    'Test if an Inspector is active
    If Not TypeName(myinspector) = "Nothing" Then
        'Display subject of active inspector
        MsgBox "The active item is " & myinspector.CurrentItem.Subject
    End If
End Sub
```
ActiveWindow Method

Returns an object representing the topmost Microsoft Outlook window on the desktop, either an Explorer or an Inspector object. If no Outlook explorer or inspector is open, returns Nothing.

expression.ActiveWindow

expression Required. An expression that returns an Application object.
Example

This Microsoft Visual Basic/Visual Basic for Applications example minimizes the topmost Microsoft Outlook window if it is an inspector window.

Dim myOlApp As New Outlook.Application
If TypeName(myOlApp.ActiveWindow) = "Inspector" Then
  myOlApp.ActiveWindow.WindowsState = olMinimized
End If

If you use VBScript, you do not create the Application object, and you cannot use named constants. This example shows how to perform the same task using VBScript.

If TypeName(Application.ActiveWindow) = "Inspector" Then
  Application.ActiveWindow.WindowsState = 1
End If
Add Method

- Add method as it applies to the Actions object.

Creates a new action in the Actions collection and returns the new action as an Action object.

expression.Add

equation  Required. An expression that returns an Actions collection object.

- Add method as it applies to the AddressEntries object.

Adds a new entry to the AddressEntries collection and returns the new entry as an AddressEntry object.

equation.Add(Type, Name, Address)

equation  Required. An expression that returns an AddressEntries object.

Type  Required String. The type of the new entry.

Name  Optional Variant. The name of the new entry.

Address  Optional Variant. The address.

- Add method as it applies to the Attachments object.

Creates a new address entry in the AddressEntries collection and returns the new address as an AddressEntry object.

Note  New entries or changes to existing entries are not persisted in the collection until after calling the Update method.

expression.Add(Source, Type, Name, Address)
expression Required. An expression that returns an **AddressEntries** collection object.

**Source** Required **String**. The source of the attachment.

**Type** Required **String**. The type of the address entry.

**Name** Optional **String**. Display name for the address entry.

**Address** Optional **String**. The e-mail address of the address entry.

- **Add method as it applies to the Explorers object.**

Creates a new instance of the Explorer window and returns a new instance of the window as an **Explorer** object.

`expression.Add(Folder, DisplayMode)`

**Folder** Required. The **Variant** object to display in the explorer window when it is created.

**DisplayMode** Optional **Long**. The display mode of the folder. Can be one of the following **OIFolderDisplayMode** constants:

- OIFolderDisplayFolderOnly
- OIFolderDisplayNoNavigation
- OIFolderDisplayNormal

**Note** The **Folder** argument can represent either a **MAPIFolder** object or the URL to that Folder.

- **Add method as it applies to the Folders object.**

Creates a new folder in the **Folders** collection, and returns the new folder as a **MAPIFolder** object.

`expression.Add(Name, Type)`
expression  Required. An expression that returns a **Folders** object.

**Name**  Required **String**. The display name for the new folder.

**Type**  Optional **Long**. The Outlook folder type for the new folder. If the folder type is not specified, the new folder will default to the same type as the folder in which it is created. Can be one of the following **OlDefaultFolders** constants: **olFolderCalendar**, **olFolderContacts**, **olFolderDrafts**, **olFolderInbox**, **olFolderJournal**, **olFolderNotes**, or **olFolderTasks**. (The constants **olFolderDeletedItems**, **olFolderOutbox**, and **olFolderSentMail** cannot be specified for this argument.)

- **Add method as it applies to the Inspectors object.**

Creates a new inspector window and returns the resulting **Inspector** object.

```
expression.Add(Item)
```

*expression*  Required. An expression that returns an **Inspectors** collection object.

**Item**  Required **Object**. The item to display in the inspector window when it is created.

- **Add method as it applies to the ItemProperties object.**

Adds an **ItemProperty** object to the **ItemProperties** collection.

```
expression.Add(Name, Type, AddToFolderFields, DisplayFormat)
```

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

**Name**  Required **String**. The name of the new item property object.

**Type**  Required **OlUserPropertyType**. The type of the new **ItemProperty**.

OlUserPropertyType can be one of these OlUserPropertyType constants: **olCombination**, **olCurrency**, **olDateTime**
olDuration
olFormula
olKeywords
olNumber
olOutlookInternal
olPercent
olText
olYesNo

**AddToFolderFields** Optional **Variant**. Determined if the **ItemProperty** will appear as a field in a folder. If **True**, the field will appear in the folder.

**DisplayFormat** Optional **Variant**. Defines the format of the field as it appears in a given folder.

▷ Add method as it applies to the **Items** object.

Creates a new Outlook item in the **Items** collection for the folder, and returns the new item. If not specified, the **Type** of the item defaults to the type of the folder, or to **MailItem** if the parent folder is not typed.

```
expression.Add(Type)
```

expression  Required. An expression that returns an **Items** collection object.

**Type**  Optional. **Variant**. The Outlook item type for the new item. Can be one of the following **OlItemType** constants: **olAppointmentItem**, **olContactItem**, **olJournalItem**, **olMailItem**, **olNoteItem**, **olPostItem**, or **olTaskItem**, one of the following **OlOfficeDocItemsType** constants: **olExcelWorkSheetItem**, **olPowerPointShowItem**, or **olWordDocumentItem**, or any valid message class. Specify the **MessageClass** to create custom forms.

▷ Add method as it applies to the **Links** object.

Links an item to another item by adding a **Link** object to the **Links** collection associated with the latter item.

```
expression.Add(Item)
```
expression  Required. An expression that returns a **Links** collection object.

**Item**  Required **Object**. The item to be linked to the item associated with the **Links** collection.

- **Add method as it applies to the OutlookBarGroups object.**

  Adds a new, empty group to the Outlook Bar and returns the new group as an **OutlookBarGroup** object.

  ```vba
  expression . Add ( Name , Index )
  ```

  **expression**  Required. An expression that returns an **OutlookBarGroups** collection object.

  **Name**  Required **String**. The name of the group being created.

  **Index**  Optional **Long**. The position at which the new group will be inserted in the Outlook Bar. Position one is at the top of the bar.

  - **Add method as it applies to the OutlookBarShortcuts object.**

  Adds a new shortcut to a group in the Outlook Bar and returns the new shortcut as an **OutlookBarShortcut** object.

  ```vba
  expression . Add ( Target , Name , Index )
  ```

  **expression**  Required. An expression that returns an **OutlookBarShortcuts** collection object.

  **Target**  Required **Variant**. The target of the shortcut being created.

  **Name**  Required **String**. The name of the shortcut being created.

  **Index**  Optional **Long**. The position at which the new shortcut will be inserted in the Outlook Bar group. Position one is at the top of the group.

  The **Target** type depends on the shortcut type. If the type is **MAPIFolder**, the shortcut represents a Microsoft Outlook folder. If the type is a **String**, the shortcut represents a file-system path or a URL.
Add method as it applies to the **Pages** object.

Creates a new page in the **Pages** collection and returns the new object. The **Pages** collection is initially empty, and there is a limit of 5 customizable pages per collection.

`expression.Add(Name)`

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

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

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

Adds a new custom property page to the Microsoft Outlook **Options** dialog box or to the folder **Properties** dialog box.

`expression.Add(Page, Title)`

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

**Page** Required **Variant.** The property page being added to the dialog box.

**Title** Optional **String.** The caption to be displayed on the property-page tab.

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

Creates a new recipient in the **Recipients** collection and returns the new recipient as a **Recipient** object.

`expression.Add(Name)`

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

**Name** Required **String.** The display name of the recipient.

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

Creates a new user property in the **UserProperties** collection, and returns the new property as a **UserProperty** object.
expression.Add(Name, Type, AddToFolderFields, DisplayFormat)

expression  Required. An expression that returns a UserProperties collection object.

Name  Required String. The name of the property.

Type  Required OlUserPropertyType. The type of the new property.

OlUserPropertyType can be one of these OlUserPropertyType constants.

olCombination
olCurrency
olDateTime
olDuration
olFormula
olKeywords
olNumber
olOutlookInternal
olPercent
olText
olYesNo

AddToFolderFields  Optional Boolean. True if the property will be added to the folder fields, False if not. The default value is True.

DisplayFormat  Optional Long. The index format of the specified OlUserPropertyType constant.

Add method as it applies to the Views object.

Creates a new view. OlViewType.

expression.Add(Name, ViewType, SaveOption)

expression  Required. An expression that returns a Views object.

Name  Required String. The name of the new view.
**ViewType**  Required **OlViewType**. The type of the new view.

OlViewType can be one of these OlViewType constants.

- **olCalendarView**
- **olCardView**
- **olIconView**
- **olTableView**
- **olTimelineView**

**SaveOption**  Optional **olViewSaveOption**. The save option that specifies the permissions of the new view.

**Note** The save option values are as follows:

- **olViewSaveOptionAllFoldersOfType** The view can be accessed in all folders of this type.
- **olViewSaveOptionThisFolderEveryOne** The view can be accessed by all users in this folder only.
- **olViewSaveOptionThisFolderOnlyMe** The view can be accessed in this folder only by the user.
Example

- As it applies to the *Explorers* object.
Remarks

The explorer window is initially hidden. You must call the **Display** of the **Explorer** object to make it visible.

The following Microsoft Visual Basic/Visual Basic for Applications example displays the Drafts folder in an explorer window without an Outlook Bar or Folder List.

```vbnet
Dim myOlApp As New Outlook.Application
Dim myExplorers As Outlook.Explorers
Dim myOlExpl As Outlook.Explorer
Dim myFolder As Outlook.MAPIFolder
Set myExplorers = myOlApp.Explorers
Set myFolder = myOlApp.GetNamespace("MAPI").GetDefaultFolder(
    olFolderDrafts)
Set myOlExpl = myExplorers.Add(
    myFolder, olFolderDisplayNoNavigation)
myOlExpl.Display
```

If you use VBScript, you do not create the **Application** object, and you cannot use named constants. This example shows how to perform the same task using VBScript.

```vbnet
Set myExplorers = Application.Explorers
Set myFolder = Application.GetNamespace("MAPI").GetDefaultFolder(16)
Set myOlExpl = myExplorers.Add(myFolder, 2)
myOlExpl.Display
```

- As it applies to the **Inspectors** object.
Remarks

This method is essentially identical to the **GetInspector** property.

This Microsoft Visual Basic/Visual Basic for Applications example prompts the user for a company name, uses the **Restrict** method to locate all contact items in the Contacts folder with that name, and displays each one.

```vbnet
Dim myOlApp As New Outlook.Application
Answer = InputBox("Enter the company name")
Set myFolder = myOlApp.GetNamespace("MAPI")_.GetDefaultFolder(olFolderContacts)
Set myItems = myFolder.Items.Restrict("[MessageClass] = 'IPM.Contact"
Set myRestrictItems = myItems.Restrict("[CompanyName] = " & Answer)
For x = 1 To myRestrictItems.Count
  Set myInspector = myOlApp.Inspectors.Add(myRestrictItems.Item(x))
  myInspector.Display
Next x

If you use VBScript, you do not create the **Application** object, and you cannot use named constants. This example shows how to perform the same task using VBScript.

```vbnet
Answer = InputBox("Enter the company name")
Set myFolder = _
  Application.GetNamespace("MAPI").GetDefaultFolder(10)
Set myItems = myFolder.Items.Restrict("[MessageClass] = 'IPM.Contact"
Set myRestrictItems = myItems.Restrict("[CompanyName] = " & Answer)
For x = 1 To myRestrictItems.Count
  Set myInspector = Application.Inspectors.Add(myRestrictItems.Item(x))
  myInspector.Display
Next x
```

- As it applies to the **Actions** object.

This Visual Basic for Applications example creates a new mail message and uses the **Add** method to add an **Action** to it.

```vbnet
Set myOlApp = CreateObject("Outlook.Application")
Set myItem = myOlApp.CreateItem(olMailItem)
Set myAction = myItem.Actions.Add
```
If you use VBScript, you do not create the `Application` object, and you cannot use named constants. This example shows how to perform the same task using VBScript.

```vb
Set myItem = Application.CreateItem(0)
Set myAction = myItem.Actions.Add
```

As it applies to the *AddressEntries* object.

This VBScript example uses the **Click** event of a **CommandButton**. The name in the To field of the form, as well as the person's manager, is located in the Global Address List. Both of these entries are added to the sender's Personal Address Book by using the **Add** method.

```vb
Sub CommandButton1_Click()
    myName = Item.To
    Set myNameSpace = Application.GetNameSpace("MAPI")
    Set myGAddressList = myNameSpace.AddressLists("Global Address Li
    Set myGEntries = myGAddressList.AddressEntries
    Set myGEntry = myGEntries(myName)
    myManager = myGEntry.Manager
    Set myGEntry2 = myGEntries(myManager)
    Set myPAddressList = myNameSpace.AddressLists("Personal Address
    Set myPEntries = myPAddressList.AddressEntries
    'Add a new AddressEntry object to the personal
    'address collection with the name, address, and
    'manager of the name in your To field.
    Set myPEntry = myPEntries.Add("Microsoft Mail Address", myName)
    myPEntry.Address = myGEntry.Address
    myPEntry.Manager = myGentry.Manager
    'Update to persist the collection.
    myPEntry.Update
    'Now add the manager's info. to
    'the Personal address collection.
    Set myPEntry2 = myPEntries.Add("Microsoft Mail Address", myManag
    myPEntry2.Address = myGEntry2.Address
    myPEntry2.Manager = myGentry2.Manager
    myPEntry2.Update
End Sub
```

As it applies to the *Links* collection.

This Microsoft Visual Basic/Visual Basic for Applications example creates a new task item, and then prompts the user for the name of a contact to link to the
item. If the contact is found, it is added to the item’s **Links** collection.

```vba
Dim myOlApp As New Outlook.Application
Dim myNameSpace As Outlook.NameSpace
Dim myFolder As Outlook.MAPIFolder
Dim myTask As Outlook.TaskItem
Dim myContact As Outlook.ContactItem
Set myTask = myOlApp.CreateItem(olTaskItem)
Set myNameSpace = myOlApp.GetNamespace("MAPI")
Set myFolder = myNameSpace.GetDefaultFolder(olFolderContacts)
tempstr = InputBox("Enter the name of the contact to link to this task")
If tempstr <> "" Then
    tempstr = ":[Full Name] = "" & tempstr & ""
    Set myItems = myFolder.Items.Restrict("[MessageClass] = 'IPM.Con"
    Set myContact = myItems.Find(tempstr)
    myTask.Links.Add myContact
myTask.Display
```

If you use VBScript, you do not create the **Application** object, and you cannot use named constants. This example shows how to perform the same task using VBScript.

```vbs
Set myTask = Application.CreateItem(3)
Set myNameSpace = Application.GetNamespace("MAPI")
Set myFolder = myNameSpace.GetDefaultFolder(10)
tempstr = InputBox("Enter the name of the contact to link to this task")
If tempstr <> "" Then
    tempstr = ":[Full Name] = "" & tempstr & ""
    Set myItems = myFolder.Items.Restrict("[MessageClass] = 'IPM.Con"
    Set myContact = myItems.Find(tempstr)
    myTask.Links.Add myContact
myTask.Display
```

- **As it applies to the Folders collection.**

This Visual Basic for Applications example uses the **Add** method to add the new folder named "My Contacts" to the current (default) Contacts folder.

```vba
Set myOlApp = CreateObject("Outlook.Application")
Set myNameSpace = myOlApp.GetNameSpace("MAPI")
Set myFolder = myNamespace.GetDefaultFolder(olFolderContacts)
Set myNewFolder = myFolder.Folders.Add("My Contacts")
```

If you use VBScript, you do not create the **Application** object, and you cannot use named constants. This example shows how to perform the same task using...
VBScript.

Set myNameSpace = Application.GetNameSpace("MAPI")
Set myFolder = myNamespace.GetDefaultFolder(10)
Set myNewFolder = myFolder.Folders.Add("My Contacts")

This Visual Basic for Applications example uses the `Add` method to add two new folders in the Tasks folder. The first folder, "Notes Folder", will contain note items. The second folder, "Contacts Folder", will contain contact items. If the folders already exist, a message box will inform the user.

Set myOlApp = CreateObject("Outlook.Application")
Set myNameSpace = myOlApp.GetNameSpace("MAPI")
Set myFolder = myNamespace.GetDefaultFolder(olFolderTasks)
On Error GoTo ErrorHandler
Set myNotesFolder = myFolder.Folders.Add("Notes Folder", olFolderNotes)
On Error GoTo ErrorHandler
Set myContactFolder = myFolder.Folders.Add("Contacts Folder", olFolderContacts)
Exit Sub
ErrorHandler:
    MsgBox "This folder already exists!"

➤ As it applies to the **Items** collection.

This Visual Basic for Applications example gets the current Contacts folder and adds a new **ContactItem** object to it.

Set myOlApp = CreateObject("Outlook.Application")
Set myNameSpace = myOlApp.GetNameSpace("MAPI")
Set myFolder = _
    myNamespace.GetDefaultFolder(olFolderContacts)
Set myItem = myFolder.Items.Add

This Visual Basic for Applications example adds a custom form to the default Tasks folder.

Set myOlApp = CreateObject("Outlook.Application")
Set myNameSpace = myOlApp.GetNameSpace("MAPI")
Set myFolder = _
    myNamespace.GetDefaultFolder(olFolderTasks)
Set myItems = myFolder.Items
Set myItem = myItems.Add("IPM.Task.myTask")

➤ As it applies to the **OutlookBarGroups** collection.
This Microsoft Visual Basic/Visual Basic for Applications example adds a group named Marketing as the second group in the Outlook Bar.

```vba
Dim myOlApp As New Outlook.Application
Dim myolBar As Outlook.OutlookBarPane
Set myolBar = myOlApp.ActiveExplorer.Panes.Item("OutlookBar")
myolBar.Contents_GROUPS.Add "Marketing", 2
```

If you use VBScript, you do not create the `Application` object. This example shows how to perform the same task using VBScript.

```vba
Set myolBar = Application.ActiveExplorer.Panes.Item("OutlookBar")
myolBar.Contents_GROUPS.Add "Marketing", 2
```

» As it applies to the `OutlookBarShortcuts` collection.

The following Microsoft Visual Basic/Visual Basic for Applications example adds a shortcut to the Microsoft home page on the Web.

```vba
Dim myOlApp As New Outlook.Application
Dim myOlBar As Outlook.OutlookBarPane
Dim myolGroup As Outlook.OutlookBarGroup
Dim myOlShortcuts As Outlook.OutlookBarShortcuts
Set myOlBar = myOlApp.ActiveExplorer.Panes.Item("OutlookBar")
Set myolGroup = myOlBar.Contents_GROUPS.Item(1)
Set myOlShortcuts = myolGroup.Shortcuts
myOlShortcuts.Add "http://www.microsoft.com", _
"Microsoft Home Page", 3
```

If you use VBScript, you do not create the `Application` object. This example shows how to perform the same task using VBScript.

```vba
Set myOlBar = _
    Application.ActiveExplorer.Panes.Item("OutlookBar")
Set myolGroup = myOlBar.Contents_GROUPS.Item(1)
Set myOlShortcuts = myolGroup.Shortcuts
myOlShortcuts.Add "http://www.microsoft.com", _
"Microsoft Home Page", 3
```

» As it applies to the `Recipients` collection.

This Visual Basic for Applications example creates a new mail message, uses the `Add` method to add "Allison Klein" as a To recipient, and displays the message.
Set myOlApp = CreateObject("Outlook.Application")
Set myItem = myOlApp.CreateItem(olMailItem)
Set myRecipient = myItem.Recipients.Add("Allison Klein")
myItem.Display

If you use VBScript, you do not create the Application object, and you cannot use named constants. This example shows how to perform the same task using VBScript.

Set myItem = Application.CreateItem(0)
Set myRecipient = myItem.Recipients.Add("Allison Klein")
myItem.Display

As it applies to the UserProperties collection.

This Visual Basic for Applications example creates a new ContactItem object and adds "LastDateSpokenWith" as a user property.

Set myOlApp = CreateObject("Outlook.Application")
Set myItem = myOlApp.CreateItem(olContactItem)
Set myUserProperty = myItem.UserProperties.
       .Add("LastDateSpokenWith", olDateTime)

This Visual Basic for Applications example creates a new ContactItem object and adds "Notes" as a user property. The Value is set by accessing the index of the new property.

Set myOlApp = CreateObject("Outlook.Application")
Set myItem = myOlApp.CreateItem(olContactItem)
Set myUserProperty = myItem.UserProperties.
       .Add("Notes", olText)
myItem.UserProperties(1).Value = "Neighbor"

As it applies to the Views object.

The following example creates a new view called New Table and stores it in a variable called objNewView.

Sub CreateView()
'Creates a new view

    Dim olApp As Outlook.Application
Dim objName As NameSpace
Dim objViews As Views
Dim objNewView As View

Set olApp = Outlook.Application
Set objName = olApp.GetNamespace("MAPI")
Set objViews = objName.GetDefaultFolder(olFolderInbox).Views
Set objNewView = objViews.Add(Name:="New Table", 
                           viewType:=olTableView, saveOption:=olViewSaveOp)

End Sub
AddMember Method

Add a new member to the specified distribution list. The distribution list contains `Recipient` objects that represent valid e-mail addresses.

`expression.AddMember(Recipient)`

`expression` Required. An expression that returns a `DistListItem` object.

`Recipient` Required. The recipient to be added to the list.
Remarks

Use the AddMembers method to add multiple members to a given recipients list.
Example

The following example creates a new **Recipient** object and adds the recipient to the distribution list. If the specified recipient is not valid, the **AddMember** method will fail.

```vba
Sub AddNewMember()
  'Adds a new member to the distribution list

  Dim olApp As Outlook.Application
  Dim objItem As DistListItem
  Dim objMail As MailItem
  Dim objRcpnt As Recipient

  Set olApp = Outlook.Application
  Set objMail = olApp.CreateItem(olMailItem)
  'Create recipient for list
  Set objRcpnt = objMail.Recipients.Add("Jeff Smith")
  Set objItem = olApp.CreateItem(olDistributionListItem)
  objItem.AddMember Recipient:=objRcpnt
  'Add note to list and display
  objItem.Body = "Regional Sales Manager - NorthWest"
  objItem.Display

End Sub
```
**AddMembers Method**

Adds new members to a distribution list.

`expression.AddMembers(Recipients)`

*expression*  Required. An expression that returns a [DistListItem](#) object.

*Recipients*  Required *Recipients* object. The members to be added to the distribution list.
Example

This Microsoft Visual Basic/Visual Basic for Applications example creates a new distribution list and adds the current user to the list.

Dim myOlApp As New Outlook.Application
Dim myNameSpace As Outlook.NameSpace
Dim myDistList As Outlook.DistListItem
Dim myTempItem As Outlook.MailItem
Dim myRecipients As Outlook.Recipients
Set myNameSpace = myOlApp.GetNamespace("MAPI")
Set myDistList = myOlApp.CreateItem(olDistributionListItem)
Set myTempItem = myOlApp.CreateItem(olMailItem)
Set myRecipients = myTempItem.Recipients
myDistList.DLName = _
    InputBox("Enter the name of the new distribution list")
myRecipients.Add myNameSpace.CurrentUser.Name
myDistList.AddMembers myRecipients
myDistList.Display

If you use VBScript, you do not create the Application object, and you cannot use named constants. This example shows how to perform the same task using VBScript.

Set myNameSpace = Application.GetNamespace("MAPI")
Set myDistList = Application.CreateItem(7)
Set myTempItem = Application.CreateItem(0)
Set myRecipients = myTempItem.Recipients
myDistList.DLName = _
    InputBox("Enter the name of the new distribution list")
myRecipients.Add myNameSpace.CurrentUser.Name
myDistList.AddMembers myRecipients
myDistList.Display
AddStore Method

Adds a personal folder (.pst) file to the current profile.

expression.AddStore(Store)

expression  Required. An expression that returns a NameSpace object.

Store  Required Variant. The path of the .pst file to be added to the profile. If the .pst file does not exist, Microsoft Outlook creates it.
**Example**

This Microsoft Visual Basic/Visual Basic for Applications example adds a new personal folder (.pst) file to the user’s profile.

```vba
Dim myOlApp As New Outlook.Application
Dim myNS As Outlook.NameSpace
Set myNS = myOlApp.GetNamespace("MAPI")
myNS.AddStore "c:\" & myNS.CurrentUser & ".pst"
```

If you use VBScript, you do not create the *Application* object. This example shows how to perform the same task using VBScript.

```vbs
Set myNS = Application.GetNamespace("MAPI")
myNS.AddStore "c:\" & myNS.CurrentUser & ".pst"
```
AddToFavorites Method

Adds the current MAPI folder to the Internet Explorer favorites list.

expression.AddToFavorites(fNoUI, Name)

eexpression  Required. An expression that returns a MAPIFolder object.

fNoUI  Optional Variant. Specifies that the folder is added without the standard user interface input box. This requires that the Name argument is specified.

Name  Optional Variant. Specifies the name of the favorite folder, if the fNoUI argument is included.
Example

The following example adds the current folder to the Favorites list in Internet Explorer. The subroutine accepts a `MAPIFolder` object and a `String` that represents the folder's name in the Favorites list. It executes the `AddToFavorites` method, using the String value supplied by the user as its argument. The `fNoUI` argument is not specified, meaning that the user interface will not appear.

Sub FaveChange()
    Dim appolApp As Outlook.Application
    Dim nmsName As NameSpace 'the namespace
    Dim fldFolder As MAPIFolder 'the folder
    Dim strName As String 'user created string
    Set appolApp = Outlook.Application
    'Create instance of namespace
    Set nmsName = appolApp.GetNamespace("Mapi")
    Set fldFolder = nmsName.GetDefaultFolder(olFolderInbox)
    'Prompt user for a Favorites list name
    strName = _
        InputBox("Enter the name of the folder as it will appear in"
    Call FaveList(fldFolder, strName)
End Sub

Sub FaveList(ByRef fldFolder As MAPIFolder, ByVal strName As String)
    'Add a Folder object to the Favorites list in Internet Explorer
    'Call method with strName as name argument
    fldFolder.AddToFavorites fNoUI:= True, Name:=strName
    'Display a message to the user
    MsgBox "The folder " & fldFolder.Name & _
        " was added to the Internet Explorer favorites list as "
End Sub
AddToPFFavorites Method

- Adds the current folder to the Public Folder's Favorites Folder.

expression.AddToPFFavorites

description Required. An expression that returns a `MAPIFolder` object.
Remarks

Only Public Folders can be added to the Favorites Folder.
Example

The following example adds the public folder Internet Newsgroups to the user's Favorites Folder by using the AddToPFFavorites method.

Sub AddToFavorites()
' Adds a Public Folder to the List of favorites

    Dim olapp As Outlook.Application
    Dim objFolder As MAPIFolder

    Set olapp = Outlook.Application
    Set objFolder = olapp.GetNamespace("mapi").Folders.Item(1).Folders(2).Folders("Internet Newsgroups")
    objFolder.AddToPFFavorites

End Sub
AdvancedSearch Method

- Performs a search based on a specified SQL search string and returns a **Search** object.

  \textit{expression}.\textbf{AdvancedSearch}(\textit{Scope}, \textit{Filter}, \textit{SearchSubFolders}, \textit{Tag})

- **expression**  Required. An expression that returns an **Application** object.

- **Scope**  Required **String**. The scope of the search. For example, the name of a folder.

- **Filter**  Optional **Variant**. The DASL search filter that defines the parameters of the search.

- **SearchSubFolders**  Optional **Variant**. Determines if the search will include any of the folder's subfolders.

- **Tag**  Optional **Variant**. The name given as an identifier for the search.
Remarks

You can run multiple searches simultaneously by calling the `AdvancedSearch` method in successive lines of code. In order to capture meaningful results, however, use the `AdvancedSearchComplete` event to signal the end of a synchronous search.
**Example**

The following example uses the **AdvancedSearch** method to create a new search. The parameters of the search, as specified by the **Filter** argument of the **AdvancedSearch** method, will return all items in the Inbox that aren't flagged.

```vba
Sub SearchForFlags()
    'List all items in the Inbox that do NOT have a flag:
    Dim objSch As Search
    Const strF As String = "urn:schemas:httpmail:messageflag = 0" & " OR urn:schemas:httpmail:messageflag IS NULL"
    Const strS As String = "Inbox"
    Set objSch = Application.**AdvancedSearch**(strS, strF)
End Sub
```

The following example returns all items in the Inbox with the subject Office Christmas Party. The **Tag** argument associates the search with a string. This is useful when conducting multiple searches.

```vba
Sub SearchInboxFolder()
    'Searches the Inbox
    Dim objSch As Search
    Const strF As String = "urn:schemas:mailheader:subject = 'Office Christmas Party'"
    Const strS As String = "Inbox"
    Const strTag As String = "SubjectSearch"
    Set objSch = Application.**AdvancedSearch**(Scope:=strS, Filter:=strF, Tag:=strTag)
End Sub
```

The following example uses the **AdvancedSearchComplete** event to capture the outcome of the search. The tag property of the search is displayed to notify the user which search has completed. This event is important because it signals the end of the search and allows you to trap a complete and meaningful set of data. This is the only way to ensure that the search has completed.

```vba
Private Sub Application_AdvancedSearchComplete(ByVal SearchObject As Search)
    Dim objRsts As Results
    MsgBox "The search " & SearchObject.Tag & " has completed."
    Set objRsts = SearchObject.Results
    'Print out number in Results collection
    Debug.Print objRsts.Count
    'Print out each member of Results collection
End Sub
```
For Each Item In objRsts
    Debug.Print Item
Next
End Sub

You can also conduct searches on multiple folders simultaneously. The following example searches for all items with the subject "Fiftieth Birthday Party" in the user's Inbox, Calendar, and Tasks folders.

Sub SearchForSubject()
    'Search for all items with a certain subject
    'in multiple folders

        Dim objSch As Search
        'Search for items where subject is not an empty string
        Const strFilter As String = _
            "urn:schemas:httpmail:subject = 'Fiftieth Birthday Party'"
        'In the Inbox, Calendar, and Tasks folders
        Const strScope As String = "'Inbox', 'Calendar', 'Tasks'"
        Set objSch = Application.AdvancedSearch(strScope, strFilter)

End Sub
Apply Method

Applies the changes that have been made in a custom property page.

expression.Apply

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

Because the PropertyPage is an abstract object that is implemented in your application (rather than by Microsoft Outlook itself), the implementation of the Apply method resembles an event procedure in your program code. That is, you write the code that implements the method in much the same way you would write an event procedure. In other words, Outlook calls the Apply method to notify your program that the user has taken an action in the dialog box displaying the custom property page that requires your program to apply the property values changed by the user.
Example

This Microsoft Visual Basic/Visual Basic for Applications example sets two global variables to reflect the values in controls on a form and then sets a global variable representing the *Dirty* property to *False*.

Private Sub PropertyPage_Apply()
    globWorkGroup = Form1.Text1.Text
    globUserType = Form1.ComboBox1.Text
    globDirty = False
End Sub
Assign Method

Assigns a task and returns a TaskItem object that represents it. This method allows a task to be assigned (delegated) to another user. You must create a task before you can assign it, and you must assign a task before you can send it. An assigned task is sent as a TaskRequestItem object.

expression.Assign

description

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

This Visual Basic for Applications example uses **CreateItem** to create a simple task and delegate it as a task request to another user.

```vba
Set myOlApp = CreateObject("Outlook.Application")
Set myItem = myOlApp.CreateItem(olTaskItem)
myItem.Assign
Set myDelegate = myItem.Recipients.Add("April LaMonte")
myItem.Subject = "Prepare Agenda For Meeting"
myItem.DueDate = #9/20/97#
myItem.Send
```

If you use VBScript, you do not create the **Application** object, and you cannot use named constants. This example shows how to perform the same task using VBScript.

```vbs
Set myItem = Application.CreateItem(3)
myItem.Assign
Set myDelegate = myItem.Recipients.Add("April LaMonte")
myItem.Subject = "Prepare Agenda For Meeting"
myItem.DueDate = #9/20/97#
myItem.Send
```
CancelResponseState Method

Resets an unsent response to a task request back to a simple task. After you receive a task request and respond to it, but before sending the response, you can use this method to revert the task to its state before you responded.

expression.CancelResponseState

expression  Required. An expression that returns a TaskItem object.
ClearConversationIndex Method

Cleans the index of the conversation thread for the mail message or post.

expression.ClearConversationIndex

expression Required. An expression that returns a MailItem or PostItem object.
ClearRecurrencePattern Method

Removes the recurrence settings and restores the single-occurrence state for an appointment or task.

_expression_.ClearRecurrencePattern

_expression_ Required. An expression that returns an AppointmentItem or TaskItem object.
Close Method

For the **Explorer** object, the **Close** method closes the explorer. No information is saved.

For an **Inspector** or Microsoft Outlook **item** object, the **Close** method closes the inspector or item and optionally saves changes to the displayed Outlook item.

`expression.Close(SaveMode)`

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

**SaveMode** This argument is used with all objects in the Applies To list except for the **Explorer** object. Required **OlInspectorClose**. The close behavior. If the item displayed within the inspector has not been changed, this argument has no effect.

**OlInspectorClose** can be one of these **OlInspectorClose** constants.

**olDiscard** Discard all changes without prompting.

**olPromptForSave** Prompt to save or discard all changes.

**olSave** Save all changes without prompting.
Example

This Visual Basic for Applications example uses **CreateItem** to open a mail message, adds a recipient to it to introduce a change, then uses the **Close** method to close the item and prompt the user to save changes.

```vba
Set myOlApp = CreateObject("Outlook.Application")
Set myItem = myOlApp.CreateItem(olMailItem)
myItem.Recipients.Add "David Goodhand"
myItem.Close olPromptForSave
```

If you use VBScript, you do not create the **Application** object, and you cannot use named constants. This example shows how to perform the same task using VBScript.

```vbs
Set myItem = Application.CreateItem(0)
myItem.Recipients.Add "David Goodhand"
myItem.Close 2
```
Show All
Copy Method

- Copy method as it applies to the View object.

Creates a new instance of a View object.

expression.Copy(Name, SaveOption)

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

Name  Required String. Represents the name of the new View object.

SaveOption  Optional OlViewSaveOption. The save option that defines the permissions of the View object.

OlViewSaveOption can be one of these OlViewSaveOption constants.

- olViewSaveOptionAllFoldersOfType
- olViewSaveOptionThisFolderEveryone
- olViewSaveOptionThisFolderOnlyMe

- Copy method as it applies to the AppointmentItem, ContactItem, DistListItem, DocumentItem, JournalItem, MailItem, MeetingItem, NoteItem, PostItem, RemoteItem, ReportItem, TaskItem, TaskRequestAcceptItem, TaskRequestDeclineItem, TaskRequestItem, and TaskRequestUpdateItem objects.

Creates another instance of an object.

expression.Copy

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

As it applies to the View object.

The following example creates a copy of a view called "New Table View" and saves it in the current folder.

Sub CopyView()
'Copies a view

    Dim olApp As Outlook.Application
    Dim objViews As Views
    Dim objNewView As View

    Set olApp = Outlook.Application
    'Create copy of View object
    Set objNewView = objViews("New Table View").Copy(Name:="Table View Copy", SaveOption:=olViewSaveOptionThisFolderEveryone)

End Sub

As it applies to the AppointmentItem, ContactItem, DistListItem, DocumentItem, JournalItem, MailItem, MeetingItem, NoteItem, PostItem, RemoteItem, ReportItem, TaskItem, TaskRequestAcceptItem, TaskRequestDeclineItem, TaskRequestItem, and TaskRequestUpdateItem objects.

This Visual Basic for Applications example creates a mail message, sets the Subject to "Speeches", uses the Copy method to copy it, then moves the copy into a newly created mail folder named "Saved Mail" within the Tasks folder.

Set myOlApp = CreateObject("Outlook.Application")
Set myNamespace = myOlApp.GetNamespace("MAPI")
Set myFolder = myNamespace.GetDefaultFolder(olFolderInbox)
Set myNewFolder = myFolder.Folders.Add("Saved Mail", olFolderDrafts)
Set myItem = myOlApp.CreateItem(olMailItem)
myItem.Subject = "Speeches"
Set myCopiedItem = myItem.Copy
myCopiedItem.Move myNewFolder
If you use VBScript, you do not create the **Application** object and you cannot use named constants. This example shows how to perform the same task using VBScript.

```vbscript
Set myNamespace = Application.GetNamespace("MAPI")
Set myFolder = myNamespace.GetDefaultFolder(6)
Set myNewFolder = myFolder.Folders.Add("Saved Mail", 16)
Set myItem = Application.CreateItem(0)
myItem.Subject = "Speeches"
Set myCopiedItem = myItem.Copy
myCopiedItem.Move myNewFolder
```
CopyFile Method

Copies a file from a specified location into Microsoft Outlook's store and returns an Object representing the copied document.

expression.CopyFile(FilePath, DestFolderPath)

expression Required. An expression that returns an Application object.

FilePath Required String. The path name of the object you want to copy.

DestFolderPath Required String. The location you want to copy the file to.
Example

The following example copies a file from the user's hard drive to a folder in the user's Inbox.

Sub Copy()
    Dim olApp As Application
    Dim docWebpg As Object
    Set olApp = Outlook.Application
    Set docWebpg = olApp.CopyFile("C:\ftp\Index.htm", "Inbox\test")
End Sub
CopyTo Method

Copies the current folder in its entirety to the destination folder. Returns a \texttt{MAPIFolder} object that represents the new copy.

\textit{expression}.\texttt{CopyTo(DestinationFolder)}

\textit{expression} Required. An expression that returns a \texttt{MAPIFolder} object (current folder).

\textit{DestinationFolder} Required \texttt{MAPIFolder} object (the destination folder for the copied folder).
Example

This Visual Basic for Applications example uses the `CopyTo` method to copy the current folder (in this case, the default Contacts folder) to the default Inbox folder.

```vba
Set myOlApp = CreateObject("Outlook.Application")
Set myNameSpace = myOlApp.GetNameSpace("MAPI")
Set myInboxFolder = myNameSpace.GetDefaultFolder(olFolderInbox)
Set myCurrentFolder = myNameSpace.GetDefaultFolder(olFolderContacts)
Set myNewFolder = myCurrentFolder.CopyTo(myInboxFolder)
```

If you use VBScript, you do not create the `Application` object, and you cannot use named constants. This example shows how to perform the same task using VBScript.

```vbs
Set myNameSpace = Application.GetNameSpace("MAPI")
Set myInboxFolder = myNameSpace.GetDefaultFolder(6)
Set myCurrentFolder = myNameSpace.GetDefaultFolder(10)
Set myNewFolder = myCurrentFolder.CopyTo(myInboxFolder)
```
CreateItem Method

Creates a new Microsoft Outlook item and returns it. The CreateItem method can only create default Outlook items. To create new items using a custom form, use the Add method on the Items collection.

expression.CreateItem(ItemType)

expression   Required. An expression that returns an Application object.

ItemType   Required OlItemType. The Outlook item Type for the new item.

OlItemType can be one of these OlItemType constants.

olAppointmentItem
olContactItem
olDistributionListItem
olJournalItem
olMailItem
olNoteItem
olPostItem
olTaskItem
Example

This Visual Basic for Applications example uses the `CreateItem` method to create a new contact in the default Contacts folder and then displays the new item.

```vba
Set myOlApp = CreateObject("Outlook.Application")
Set myNameSpace = myOlApp.GetNameSpace("MAPI")
Set myFolder = myNameSpace.GetDefaultFolder(olFolderContacts)
Set myItem = myOlApp.CreateItem(olContactItem)
myItem.Display
```

If you use VBScript, you do not create the `Application` object, and you cannot use named constants. This example shows how to perform the same task using VBScript.

```vbs
Set myNameSpace = Application.GetNameSpace("MAPI")
Set myFolder = myNameSpace.GetDefaultFolder(10)
Set myItem = Application.CreateItem(2)
myItem.Display
```
Show All
CreateItemFromTemplate Method

Creates a new Microsoft Outlook item from an Outlook template (.oft) and returns the new item.

expression.CreateItemFromTemplate(TemplatePath, InFolder)

expression  Required. An expression that returns an Application object.

TemplatePath  Required String. The path and file name of the Outlook template for the new item.

InFolder  Optional Variant. The folder in which the item is to be created. If this argument is omitted, the default folder for the item type will be used.
Example

This Visual Basic for Applications example uses `CreateItemFromTemplate` to create a new item from an Outlook template and then displays it.

```vba
Set myOlApp = CreateObject("Outlook.Application")
Set myItem = myOlApp.CreateItemFromTemplate _
    ("C:\Program Files\Outlook\While You Were Out.oft")
myItem.Display
```

If you use VBScript, you do not create the `Application` object. This example shows how to perform the same task using VBScript.

```vbs
Set myItem = Application.CreateItemFromTemplate _
    ("C:\Program Files\Outlook\While You Were Out.oft")
myItem.Display
```
Show All
CreateObject Method

Creates an Automation object of the specified class. If the application is already running, **CreateObject** will create a new instance.

This method is provided so that other applications can be automated from VBScript. It should not be used to automate Outlook from VBScript.

**Note**  The **CreateObject** methods commonly used in the example code within this Help file (available when you click "Example") are made available by Microsoft Visual Basic (VB) or Microsoft Visual Basic for Applications. These examples do not use the same **CreateObject** method that is implemented as part of Outlook's object model.

```vbscript
expression.CreateObject(ObjectName)
```

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

**ObjectName**  Required **String**. The class name of the object to create. For information about valid class names, see [OLE Programmatic Identifiers](#).
Example

This VBScript example utilizes the Open event of the Item to access Microsoft Internet Explorer and display the Microsoft Web page.

Sub Item_Open()
    Set Web = CreateObject("InternetExplorer.Application")
    Web.Visible = TRUE
    Web.Navigate "www.microsoft.com"
End Sub

This VBScript example utilizes the Click event of a CommandButton control on the Item to access Microsoft Word and open a document on the root directory named "Resume.doc".

Sub CommandButton1_Click()
    Set Word = CreateObject("Word.Application")
    Word.Visible = TRUE
    Word.Documents.Open("C:\Resume.doc")
End Sub
CreateRecipient Method

Creates and returns a `Recipient` object. This method is most commonly used to create a `Recipient` object for use with the `GetSharedDefaultFolder` method (for example, to open a delegator's folder). It can also be used to verify a given name against an address book.

```
expression.CreateRecipient(RecipientName)
```

**expression**  Required. An expression that returns a `NameSpace` object.

**RecipientName**  Required `String`. The display name of the recipient.
Example

This Visual Basic for Applications example uses the `CreateRecipient` method to obtain the default Calendar folder for the user Eric Lang.

```vba
Set myOlApp = CreateObject("Outlook.Application")
Set myNameSpace = myOlApp.GetNameSpace("MAPI")
Set myRecipient = myNameSpace.CreateRecipient("Eric Lang")
Set hisCalendar = myNameSpace.GetSharedDefaultFolder_(myRecipient, olFolderCalendar)
```

If you use VBScript, you do not create the `Application` object, and you cannot use named constants. This example shows how to perform the same task using VBScript.

```vbs
Set myNameSpace = Application.GetNameSpace("MAPI")
Set myRecipient = myNameSpace.CreateRecipient("Eric Lang")
Set hisCalendar = myNameSpace.GetSharedDefaultFolder_(myRecipient, 9)
```
Delete Method

- Deletes the object from the collection.

\textit{expression.Delete}

\textit{expression} Required. An expression that returns one of the objects in the Applies To list.
Example

This Visual Basic for Applications example uses the `GetFirst` method to locate the first folder in the Tasks folder and then uses the `Delete` method to delete the folder.

Set OlApp = CreateObject("Outlook.Application")
Set olName = OlApp.GetNameSpace("MAPI")
Set olFolder = olName.GetDefaultFolder(olFolderTasks)
Set olOldFolder = olFolder.Folders.GetFirst
olOldFolder.Delete

If you use VBScript, you do not create the `Application` object, and you cannot use named constants. This example shows how to perform the same task using VBScript.

Set olNameSpace = Application.GetNameSpace("MAPI")
Set olOldFolder = olFolder.Folders.GetFirst olOldFolder.Delete
Set olFolder = myNameSpace.GetDefaultFolder(13)
Details Method

The **Details** method displays a modal dialog box that provides detailed information about an **AddressEntry** object. You must use error handling to handle run time errors when the user clicks **Cancel** in the dialog box. The **Details** method actually stops the code from running while the dialog box is displayed.

**Note** The **Details** method fails if the **Name** property is empty.

`expression.Detai ls(HWnd)`

- `expression` Required **AddressEntry** object.
- `HWnd` Optional **Variant**. The parent window handle for the details dialog box. A zero value (the default) specifies a modal dialog box.
Example

This Visual Basic for Applications example uses the Add method to add a new AddressEntry to the Personal Address Book. The Name, "John Q. Public" is necessary to avoid errors.

If this entry already exists, the code skips to DialogBox:. The Details method displays a dialog box that shows the information for this entry.

Set myOlApp = CreateObject("Outlook.Application")
    Set myNamespace = myOlApp.GetNamespace("MAPI")
    Set myAddrList = myNamespace.AddressLists("Personal Address Book")
    Set myAddrEntries = myAddrList.AddressEntries
    Set myEntry = myAddrEntries.Add("Microsoft Mail Address")
    myEntry.Name = "John Q. Public"
    On Error GoTo DialogBox
    myEntry.Address = "someone@microsoft.com"
    myEntry.Update
    DialogBox:
        myEntry.Details

---
Dial Method

Displays the New Call dialog box that allows users to dial the primary phone number of a specified contact.

expression.Dial(ContactItem)

expression  Required. An expression that returns a NameSpace object.

ContactItem  Optional Variant. The ContactItem object of the contact you want to dial.
Example

The following example opens the **New Call** dialog box.

```vba
Sub DialContact()
 'Opens the New Call dialog

    Dim olApp As Outlook.Application
    Set olApp = Outlook.Application
    olApp.GetNamespace("Mapi").Dial

End Sub
```

The following example opens the **New Call** dialog box with the Contact's information.

```vba
Sub DialContact()
 'Opens the New Call dialog with the contact info

    Dim olApp As Outlook.Application
    Dim objContact As ContactItem

    Set olApp = Outlook.Application
    Set objContact = olApp.GetNamespace("Mapi").GetDefaultFolder(olFolderContacts).Items("Jeff Smith")
    olApp.GetNamespace("Mapi").Dial objContact

End Sub
```
Dismiss Method

- Dismisses the current reminder.

*expression*.**Dismiss**

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

The **Dismiss** method will fail if there is no visible reminder.
Example

The following example dismisses all active reminders. A reminder is active if its `IsVisible` property is set to `True`.

Sub DismissReminders()
'Dismisses any active reminders

    Dim olApp As Outlook.Application
    Dim objRems As Reminders
    Dim objRem As Reminder

    Set olApp = Outlook.Application
    Set objRems = olApp.Reminders

    For Each objRem In objRems
        If objRem.IsVisible = True Then
            objRem.Dismiss
        End If
    Next objRem

End Sub
Display Method

For an Explorer or MAPIFolder object, the Display method displays a new Explorer object for the folder.

For specified Microsoft Outlook items, the Display method displays a new Inspector object for the item.

**Note** The Display method is supported for explorer and inspector windows for the sake of backward compatibility. To activate an explorer and inspector window, use the Activate method.

`expression.Display(Modal)`

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

*Modal*  This argument is used with all objects in the Applies To list except for the Explorer and MAPIFolder objects. Optional Variant. True to make the Inspector object modal. The default value is False.
Example

This Visual Basic for Applications example uses the **Display** method to display the default Inbox folder. This example will not return an error, even if there are no items in the Inbox, because you are not asking for the display of a specific item.

```vba
Set myOlApp = CreateObject("Outlook.Application")
Set myNameSpace = myOlApp.GetNameSpace("MAPI")
Set myFolder = myNameSpace.GetDefaultFolder(olFolderInbox)
myFolder.Display
```

If you use VBScript, you do not create the **Application** object, and you cannot use named constants. This example shows how to perform the same task using VBScript.

```vbs
Set myNameSpace = Application.GetNameSpace("MAPI")
Set myFolder = myNameSpace.GetDefaultFolder(6)
myFolder.Display
```

This Visual Basic for Applications example displays the first item in the Inbox folder. This example will return an error if the Inbox is empty, because you are attempting to display a specific item. If there are no items in the folder, a message box will be displayed to inform the user.

```vba
Set myOlApp = CreateObject("Outlook.Application")
Set myNameSpace = myOlApp.GetNameSpace("MAPI")
Set myFolder = myNameSpace.GetDefaultFolder(olFolderInbox)
On Error GoTo ErrorHandler
myFolder.Items(1).Display
Exit Sub
ErrorHandler:
    MsgBox "There are no items to display! "
```
Execute Method

Executes the action for the specified item. Returns the Microsoft Outlook item created by the action.

expression.Execute

expression  Required. An expression that returns an Action object.
Example

This Visual Basic for Applications example uses the **Execute** method to look through all the actions for the given mail message, and executes the action called "Reply to All."

```vba
Set myOlApp = CreateObject("Outlook.Application")
Set myNameSpace = myOlApp.GetNameSpace("MAPI")
On Error GoTo ErrorHandler
Set myItem = myOlApp.ActiveInspector.CurrentItem
For Each a In myItem.Actions
    If a.Name = "Reply to All" Then
        Set myItem2 = a.Execute
        Exit For
    End If
Next a
Exit Sub
ErrorHandler:
    MsgBox "There is no current item!"
```
Find Method

- Find method as it applies to the Items object.

Locates and returns an item.

expression.Find(Filter)

expression  Required. An expression that returns an Items object.

Filter  Required String. The filter of the search.

- Find method as it applies to the UserProperties object.

Locates and returns a UserProperty object for the requested property name, if it exists.

expression.Find(Name, Custom)

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

Name  Required String. The name of the requested property.

Custom  Optional Variant. A custom string that defines the search parameters.
Example

This Visual Basic for Applications example finds a custom property named "LastDateContacted" for the contact.

Sub FindContact()
    'Finds and displays last contacted info for a contact

    Dim olApp As Outlook.Application
    Dim objContact As ContactItem
    Dim objContacts As MAPIFolder
    Dim objNameSpace As NameSpace
    Dim objProperty As UserProperty

    Set olApp = CreateObject("Outlook.Application")
    Set objNameSpace = olApp.GetNamespace("MAPI")
    Set objContacts = objNameSpace.GetDefaultFolder(olFolderContacts)
    Set objContact = objContacts.Items.Find("[FileAs] = ""Smith, Jeff"

    If Not TypeName(objContact) = "Nothing" Then
        Set objProperty = objContact.UserProperties.Find("LastDateContacted")
        If TypeName(objProperty) <> "Nothing" Then
            MsgBox "Last Date Contacted: " & objProperty.Value
        End If
    Else
        MsgBox "Contact not found."
    End If
End Sub
Show All
FindNext Method

After the Find method runs, this method finds and returns the next Microsoft Outlook item in the specified collection. The search operation begins from the current position, which matches the expression previously set through the Find method.

expression.FindNext

type  Required. An expression that returns an Items collection object.
Example

This Visual Basic for Applications example uses the `GetDefaultFolder` method to return the `MAPIFolder` object that represents the default Calendar folder for the current user. It then uses the `Find` and `FindNext` methods to locate all the appointments that occur today and display them in a series of message boxes.

```vba
Set myOlApp = CreateObject("Outlook.Application")
Set myNameSpace = myOlApp.GetNameSpace("MAPI")
tdystart = Format(Date, "Short Date") & " 12:00 AM"
tdyend = Format(Date, "Short Date") & " 11:59 PM"
Set myAppointments = myNameSpace.GetDefaultFolder(olFolderCalendar).Items
Set currentAppointment = myAppointments.Find("[Start] >= " & _
    tdystart & "" & [Start] <= "" & tdyend & """)
While TypeName(currentAppointment) <> "Nothing"
    MsgBox currentAppointment.Subject
    Set currentAppointment = myAppointments.FindNext
Wend
```
Show All
Forward Method

Executes the Forward action for an item. Returns the resulting copy as a new object (as a MeetingItem object for the MeetingItem object, or a MailItem object for all other objects in the Applies To list).

expression.Forward

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

This Visual Basic for Applications example uses the **GetDefaultFolder** method to return the **MAPIFolder** object that represents the default Inbox folder for the current user. It then uses the **Forward** method to retrieve the first message in the default Inbox folder and forward it to Laura Jennings. The **Add** method is used to add Laura Jennings to the **Recipients** collection and the **Send** method sends the **item** to all recipients. It is assumed that the name will resolve unambiguously in the Address Book.

```vba
Set myOlApp = CreateObject("Outlook.Application")
Set myNameSpace = myOlApp.GetNameSpace("MAPI")
Set myFolder = myNameSpace.GetDefaultFolder(olFolderInbox)
Set myForward = myFolder.Items(1).Forward
myForward.Recipients.Add "Laura Jennings"
myForward.Send
```

If you use VBScript, you do not create the **Application** object, and you cannot use named constants. This example shows how to perform the same task using VBScript.

```vbs
Set myNameSpace = Application.GetNameSpace("MAPI")
Set myFolder = myNameSpace.GetDefaultFolder(6)
Set myForward = myFolder.Items(1).Forward
myForward.Recipients.Add "Laura Jennings"
myForward.Send
```
ForwardAsVcal Method

Forwards the AppointmentItem as a vCal; virtual calendar item. The ForwardAsVcal method returns a MailItem with the vCal file attached.

expression.ForwardAsVcal

expression Required. An expression that returns an AppointmentItem object.
ForwardAsVcard Method

Forwards the **ContactItem** as a vCard, the Internet standard for creating and sharing virtual business cards. The **ForwardAsVcard** method returns a **MailItem** with the vCard file attached.

*expression*.**ForwardAsVcard**

*expression* Required. An expression that returns a **ContactItem** object.
FreeBusy Method

Returns free/busy information for the recipient. The default is to return a string representing one month of free/busy information compatible with the Microsoft Schedule+ Automation format (that is, the string contains one character for each \textit{MinPerChar} minute, up to one month of information from the specified \textit{Start} date).

If the optional argument \textit{CompleteFormat} is omitted or \texttt{False}, then "free" is indicated by the character 0 and all other states by the character 1.

If \textit{CompleteFormat} is \texttt{True}, then the same length string is returned as defined above, but the characters now correspond to the \texttt{OlBusyStatus} constants: \texttt{olBusy}, \texttt{olFree}, \texttt{olOutOfOffice}, or \texttt{olTentative}.

\begin{verbatim}
expression.FreeBusy(Start, MinPerChar, CompleteFormat)
\end{verbatim}

\begin{itemize}
  \item \textit{expression} Required. An expression that returns a \texttt{Recipient} object.
  \item \textit{Start} Required \texttt{Date}. The start date for the returned period of free/busy information.
  \item \textit{MinPerChar} Required \texttt{Long}. The number of minutes per character represented in the returned free/busy string.
  \item \textit{CompleteFormat} Optional \texttt{Variant}. \texttt{True} if the returned string should contain not only free/busy information, but also values for each character according to the \texttt{OlBusyStatus} constants: \texttt{olBusy}, \texttt{olFree}, \texttt{olOutOfOffice}, and \texttt{olTentative}.
\end{itemize}
Example

This Visual Basic for Applications example uses the `FreeBusy` method to return a string of free/busy information with one character for each day. This example allows for the possibility that the free/busy information for this recipient is not accessible.

```vba
Public Sub cmdCreate_Click()
    Set myOlApp = CreateObject("Outlook.Application")
    Set myNameSpace = myOlApp.GetNamespace("MAPI")
    Set myRecipient = myNameSpace.CreateRecipient("Paul Borm")
    On Error GoTo ErrorHandler
    myFBInfo = myRecipient.FreeBusy(#8/1/97#, 60 * 24)
    Exit Sub
    ErrorHandler: MsgBox "Cannot access information."
End Sub
```

This Visual Basic for Applications example returns a string of free/busy information with one character for each hour (complete format).

```vba
Set myRecipient = myNameSpace.CreateRecipient("Kevin Yim")
myFBInfo = myRecipient.FreeBusy(#8/1/97#, 60, True)
```
GetAssociatedAppointment Method

Returns an AppointmentItem object that represents the appointment associated with the meeting request.

expression.GetAssociatedAppointment(AddToCalendar)

expression  Required. An expression that returns a MeetingItem object.

AddToCalendar  Required Boolean. True to add the meeting to the default Calendar folder.
Example

This Visual Basic for Applications example finds a MeetingItem in the default Inbox folder and adds the associated appointment to the Calendar folder.

```vba
Set myOlApp = CreateObject("Outlook.Application")
Set myNameSpace = myOlApp.GetNameSpace("MAPI")
Set myFolder = _
    myNameSpace.GetDefaultFolder(olFolderInbox)
Set myMtgReq = myFolder.Items.Find _
    ("[MessageClass] = 'IPM.Schedule.Meeting.Request'")
If TypeName(myMtgReq) <> "Nothing" Then
    Set myAppt = myMtgReq.GetAssociatedAppointment(True)
End If
```

If you use VBScript, you do not create the Application object, and you cannot use named constants. This example shows how to perform the same task using VBScript.

```vbs
Set myNameSpace = Application.GetNameSpace("MAPI")
Set myFolder = _
    myNameSpace.GetDefaultFolder(6)
Set myMtgReq = myFolder.Items.Find _
    ("[MessageClass] = 'IPM.Schedule.Meeting.Request'")
If TypeName(myMtgReq) <> "Nothing" Then
    Set myAppt = myMtgReq.GetAssociatedAppointment(True)
End If
```

This example accepts a meeting request, sending the response without displaying the inspector.

```vbs
Set myNewMeeting = myMtgReq.GetAssociatedAppointment(True)
myNewMeeting.Respond olResponseAccepted, True
```
GetAssociatedTask Method

Returns a TaskItem object that represents the requested task.

**Note** The GetAssociatedTask method will not work unless the TaskItem is processed before the method is called. To do so, call the Display method before calling GetAssociatedTask.

expression.GetAssociatedTask(AddToTaskList)

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

AddToTaskList Required Boolean. True if the task is added to the default Tasks folder.
Example

This example accepts a **TaskRequestItem**, sending the response without displaying the inspector.

```vbscript
Set myolapp = CreateObject("Outlook.Application")
Set myNameSpace = myolapp.GetNamespace("MAPI")
Set myTasks = myNameSpace.GetDefaultFolder(olFolderInbox)
Set mytaskreqItem = myTasks.items.Find(_
    "[Subject] = ""Meeting w/ Jerry Wheeler"")
If Not TypeName(mytaskreqItem) = "Nothing" Then
    'The task is displayed.
    mytaskreqItem.Display
    Set myNewTaskItem = mytaskreqItem.GetAssociatedTask(True)
    myNewTaskItem.Respond olTaskAccept, True, True
End If

If you use VBScript, you do not create the **Application** object, and you cannot use named constants. This example shows how to perform the same task using VBScript.

```vbscript
Set myNameSpace = Application.GetNamespace("MAPI")
Set myTasks = myNameSpace.GetDefaultFolder(6)
Set myTaskReqItem = myTasks.items.Find("[Subject] = ""Meeting w/ Jer"
If Not TypeName(myTaskReqItem) = "Nothing" Then
    'The task is displayed.
    myTaskReqItem.Display
    Set myNewTaskItem = myTaskReqItem.GetAssociatedTask(True)
    myNewTaskItem.Respond 2, True, True
End If
```
GetDefaultFolder Method

Returns a **MAPIFolder** object that represents the default folder of the requested type for the current profile (for example, obtaining the default Calendar folder for the user who is currently logged on).

**Note** To return a specific non-default folder, use the **Folders** collection.

`expression.GetDefaultFolder(FolderType)`

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

*FolderType*  Required **OlDefaultFolders**. The type of default folder to return.

OlDefaultFolders can be one of these OlDefaultFolders constants.

*olFolderCalendar*
*olFolderContacts*
*olFolderDeletedItems*
*olFolderDrafts*
*olFolderInbox*
*olFolderJournal*
*olFolderNotes*
*olFolderOutbox*
*olFolderSentMail*
*olFolderTasks*
*olPublicFoldersAllPublicFolders*
Example

This Visual Basic for Applications example uses the **GetDefaultFolder** method to obtain the default Calendar folder for the current user.

```
Set myOlApp = CreateObject("Outlook.Application")
Set myNameSpace = myOlApp.GetNameSpace("MAPI")
Set myCalendar = myNameSpace.GetDefaultFolder(olFolderCalendar)
```

If you use VBScript, you do not create the **Application** object, and you cannot use named constants. This example shows how to perform the same task using VBScript.

```
Set myNameSpace = Application.GetNameSpace("MAPI")
Set myCalendar = myNameSpace.GetDefaultFolder(9)
```

This Visual Basic for Applications example returns the first folder in the Tasks Folders collection.

```
Set myOlApp = CreateObject("Outlook.Application")
Set myNameSpace = myOlApp.GetNameSpace("MAPI")
Set myTasks = myNameSpace.GetDefaultFolder(olFolderTasks)
Set myFolder = myTasks.Folders(1)
```
GetExplorer Method

Returns an **Explorer** object that represents a new, inactive **Explorer** object initialized with the specified folder as the current folder. This method is useful for returning a new **Explorer** object in which to display the folder, as opposed to using the **ActiveExplorer** method and setting the **CurrentFolder** property.

The **Display** method can be used to activate or show the **Explorer**.

The **GetExplorer** method takes an optional argument of an **OlFolderDisplayMode** constant.

By default, the new **Explorer** will be displayed in the **Normal** mode (**olFolderDisplayNormal**) with all interface elements displayed: a message panel on the right, an Outlook bar on the left, and a folder banner across the top from which the user can pull down a folder navigation panel. The exception to this rule is when you are calling **GetExplorer** on delegated folders which are in **No-Navigation** mode by default. You can apply more restrictions to a default mode, but you cannot lessen the restrictions by changing the **OlFolderDisplayMode**.

The explorer can also be displayed in **Folder-Only** mode (**olFolderDisplayFolderOnly**) in which the **Explorer** will display with no folder list and no Outlook bar, but the drop-down list and the navigation commands will still be available.

The most restrictive mode you can use is **No-Navigation** mode (**olFolderDisplayNoNavigation**). In this mode, the **Explorer** will display with no folder list, no drop-down folder list, and any "Go"-type menu/command bar options should be disabled. Basically, the user should not be able to navigate to any other folder within that **Explorer** window. By default, a delegated (shared) folder appears in **No-Navigation** mode.

```expression.GetExplorer(DisplayMode)```
**expression**  Required. An expression that returns a **MAPIFolder** object.

**DisplayMode**  Optional Variant. The display mode of the folder. Can be one of the following **OlFolderDisplayMode** constants: **olFolderDisplayFolderOnly**, **olFolderDisplayNoNavigation** or **olFolderDisplayNormal** (default).
**Example**

This Visual Basic for Applications example uses the `GetExplorer` method to return a new, inactive `Explorer` for `myFolder`, and then displays it in the default mode of `olFolderDisplayNormal`.

```vba
Set myOlApp = CreateObject("Outlook.Application")
Set myNameSpace = myOlApp.GetNameSpace("MAPI")
Set myFolder = myNameSpace.GetDefaultFolder(olFolderInbox)
Set myExplorer = myFolder.GetExplorer
myExplorer.Display
```

If you use VBScript, you do not create the `Application` object, and you cannot use named constants. This example shows how to perform the same task using VBScript.

```vbs
Set myNameSpace = Application.GetNameSpace("MAPI")
Set myFolder = myNameSpace.GetDefaultFolder(6)
Set myExplorer = myFolder.GetExplorer
myExplorer.Display
```
**GetFirst Method**

The **GetFirst** method returns the first object in the specified collection. Returns **Nothing** if no first object exists, for example, if there are no objects in the collection.

**Note** To ensure correct operation of the **GetFirst**, **GetLast**, **GetNext**, and **GetPrevious** methods in a large collection, call **GetFirst** before calling **GetNext** on that collection, and call **GetLast** before calling **GetPrevious**. To ensure that you are always making the calls on the same collection, create an explicit variable that refers to that collection before entering the loop.

`expression.GetFirst`

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

This Visual Basic for Applications example uses the **GetFirst** method to locate the first folder in the Tasks folder and then uses the **Delete** method to delete the folder.

```vba
Set myOlApp = CreateObject("Outlook.Application")
Set myNameSpace = myOlApp.GetNameSpace("MAPI")
Set myFolder = myNameSpace.GetDefaultFolder(olFolderTasks)
Set myOldFolder = myFolder.Folders.GetFirst
myOldFolder.Delete
```

If you use VBScript, you do not create the **Application** object, and you cannot use named constants. This example shows how to perform the same task using VBScript.

```vbs
Set myNameSpace = Application.GetNamespace("MAPI")
Set myFolder = myNameSpace.GetDefaultFolder(13)
Set myOldFolder = myFolder.Folders.GetFirst
myOldFolder.Delete
```
GetFolderFromID Method

Returns a **MAPIFolder** object identified by the specified entry ID (if valid). This method is used for ease of transition between MAPI and OLE/Messaging applications and Microsoft Outlook.

`expression.GetFolderFromID(EntryIDFolder, EntryIDStore)`

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

**EntryIDFolder**  Required **String**. The **EntryID** of the folder.

**EntryIDStore**  Optional **Variant**. The **StoreID** for the folder.
Example

This Visual Basic for Applications example obtains the **EntryID** and **StoreID** for a subfolder within the Tasks **Folders** collection and then calls the **GetFolderFromID** method using these values to obtain the same subfolder. The folder is then displayed.

```vba
Set myolapp = CreateObject("Outlook.Application")
Set myNameSpace = myolapp.GetNamespace("MAPI")
Set myFolder = myNameSpace.GetDefaultFolder(olFolderTasks)
myEntryID = myFolder.EntryID
myStoreID = myFolder.StoreID
Set myFolder = myNameSpace.GetFolderFromID(myEntryID, myStoreID)
myFolder.Display
```

If you use VBScript, you do not create the **Application** object, and you cannot use named constants. This example shows how to perform the same task using VBScript.

```vba
Set myNameSpace = Application.GetNamespace("MAPI")
Set myFolder = myNameSpace.GetDefaultFolder(13)
myEntryID = myFolder.EntryID
myStoreID = myFolder.StoreID
Set myNewFolder = myNameSpace.GetFolderFromID(myEntryID, myStoreID)
myNewFolder.Display
```
GetFreeBusy Method

The GetFreeBusy method returns a String representing the availability of the individual user for a period of 30 days from the start date, beginning at midnight of the date specified.

**Note** If an address entry represents a distribution list, the status of its individual members cannot be returned to you with the GetFreeBusy method. A meeting request should be sent only to single messaging users. You can determine if a messaging user is a distribution list by determining if its DisplayType property is olDistList or olPrivateDistList.

```vba
expression.GetFreeBusy(Start, MinPerChar, CompleteFormat)
```

*expression* Required. The AddressEntry object.

**Start** Required **Date**. Specifies the date.

**MinPerChar** Required **Long**. Specifies the length of each time slot in minutes. Default is 30 minutes.

**CompleteFormat** Optional **Variant**.
GetItemFromID Method

Returns a Microsoft Outlook item identified by the specified entry ID (if valid). This method is used for ease of transition between MAPI and OLE/Messaging applications and Outlook.

expression.GetItemFromID(EntryIDItem, EntryIDStore)

expression Required. An expression that returns a NameSpace object.

EntryIDItem Required String. The EntryID of the item.

EntryIDStore Optional Variant. The StoreID for the folder.
Remarks

*EntryIDStore* usually must be provided when retrieving an item based on its MAPI IDs.
GetLast Method

The GetLast method returns the last object in the specified collection. It returns Nothing if no last object exists, for example, if the collection is empty.

Note To ensure correct operation of the GetFirst, GetLast, GetNext, and GetPrevious methods in a large collection, call GetFirst before calling GetNext on that collection, and call GetLast before calling GetPrevious. To ensure that you are always making the calls on the same collection, create an explicit variable that refers to that collection before entering the loop.

expression.GetLast

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

Returns a Recipient object representing a member in a distribution list.

expression.GetMember(Index)

expression  Required. An expression that returns a DistListItem object.

Index  Required Long. The index number of the member to be retrieved.
Example

This Microsoft Visual Basic/Visual Basic for Applications example locates every distribution list in the default Contacts folder and determines whether the list contains the current user.

```vba
Dim myOlApp As New Outlook.Application
Dim myNameSpace As Outlook.NameSpace
Dim myFolder As Outlook.MAPIFolder
Dim myDistList As Outlook.DistListItem
Set myNameSpace = myOlApp.GetNamespace("MAPI")
Set myFolder = myNameSpace.GetDefaultFolder(olFolderContacts)
For x = 1 To myFolder.Items.Count
    If TypeName(myFolder.Items.Item(x)) = "DistListItem" Then
        Set myDistList = myFolder.Items.Item(x)
        For y = 1 To myDistList.MemberCount
            If myDistList.GetMember(y).Name = myNameSpace.CurrentUser.Name Then
                MsgBox "Your are a member of " & myDistList.DLName
            End If
        Next y
    End If
Next x
```

If you use VBScript, you do not create the Application object, and you cannot use named constants. This example shows how to perform the same task using VBScript.

```vbs
Set myNameSpace = Application.GetNamespace("MAPI")
Set myFolder = myNameSpace.GetDefaultFolder(10)
For x = 1 To myFolder.Items.Count
    If TypeName(myFolder.Items.Item(x)) = "DistListItem" Then
        Set myDistList = myFolder.Items.Item(x)
        For y = 1 To myDistList.MemberCount
            If myDistList.GetMember(y).Name = myNameSpace.CurrentUser.Name Then
                MsgBox "Your are a member of " & myDistList.DLName
            End If
        Next y
    End If
Next x
```
GetNext Method

The **GetNext** method returns the next object in the specified collection. It returns **Nothing** if no next object exists, for example, if already positioned at the end of the collection.

**Note** To ensure correct operation of the **GetFirst**, **GetLast**, **GetNext**, and **GetPrevious** methods in a large collection, call **GetFirst** before calling **GetNext** on that collection, and call **GetLast** before calling **GetPrevious**. To ensure that you are always making the calls on the same collection, create an explicit variable that refers to that collection before entering the loop.

`expression.GetNext`

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

The GetOccurrence method returns a specific instance of the AppointmentItem object on the specified date.

**Note** The GetOccurrence method generates an error if no appointment of that series exists on the specified date.

expression.GetOccurrence(StartDate)

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

StartDate  Required Date that represents local time.
Example

This Visual Basic for Applications example uses CreateItem to create an AppointmentItem object. The RecurrencePattern is obtained for this item using the GetRecurrencePattern method. By setting the RecurrencePattern properties, RecurrenceType, PatternStartDate, and PatternEndDate, the appointments are now a recurring series that occur on a daily basis for the period of one year.

An Exception object is created when one instance of this recurring appointment is obtained using the GetOccurrence method and properties for this instance are altered. This exception to the series of appointments is obtained using the GetRecurrencePattern method to access the Exceptions collection associated with this series. Message boxes display the original Subject and OriginalDate for this exception to the series of appointments and the current date, time, and subject for this exception.

For a description of changes required for this example to work in VBScript, see the Note at the end of the example.

Public Sub cmdExample()
    Set myOlApp = New Outlook.Application
    Set myApptItem = myOlApp.CreateItem(olAppointmentItem)
    myApptItem.Start = #2/2/98 3:00:00 PM#
    myApptItem.End = #2/2/98 4:00:00 PM#
    myApptItem.Subject = "Meet with Boss"

    'Get the recurrence pattern for this appointment
    'and set it so that this is a daily appointment
    'that begins on 2/2/98 and ends on 2/2/99
    'and save it.
    Set myRecurrPatt = myApptItem.GetRecurrencePattern
    myRecurrPatt.RecurrenceType = olRecursDaily
    myRecurrPatt.PatternStartDate = #2/2/98#
    myRecurrPatt.PatternEndDate = #2/2/99#
    myApptItem.Save

    'Access the items in the Calendar folder to locate
    'the master AppointmentItem for the new series.
    Set myNamespace = myOlApp.GetNamespace("MAPI")
    Set myFolder = myNamespace.GetDefaultFolder(olFolderCalendar)
Set myItems = myFolder.Items
Set myApptItem = myItems("Meet with Boss")

'Get the recurrence pattern for this appointment
'and obtain the occurrence for 3/12/98.
myDate = #3/12/98 3:00:00 PM#
Set myRecurrPatt = myApptItem.GetRecurrencePattern
Set myOddApptItem = myRecurrPatt.GetOccurrence(myDate)

'Save the existing subject. Change the subject and
'starting time for this particular appointment
'and save it.
saveSubject = myOddApptItem.Subject
myOddApptItem.Subject = "Meet NEW Boss"
newDate = #3/12/98 3:30:00 PM#
myOddApptItem.Start = newDate
myOddApptItem.Save

'Get the recurrence pattern for the master
'AppointmentItem. Access the collection of
'exceptions to the regular appointments.
Set myRecurrPatt = myApptItem.GetRecurrencePattern
Set myException = myRecurrPatt.Exceptions.Item(1)

'Display the original date, time, and subject
'for this exception.
MsgBox myException.OriginalDate & ": " & saveSubject

'Display the current date, time, and subject
'for this exception.
MsgBox myException.AppointmentItem.Start & ": " &
myException.AppointmentItem.Subject
End Sub

Note  For this example to work properly in VBScript, there are only a few
simple changes that need to be made in the code.

You don't have to retrieve the application as an object and you must use the
values of the constants, so:

Set myOlApp = New Outlook.Application
    Set myApptItem = myOlApp.CreateItem(olAppointmentItem)

becomes:

Set myApptItem = Application.CreateItem(1)
and

myRecurrPatt.RecurrenceType = olRecursDaily

becomes:

myRecurrPatt.RecurrenceType = 0

and

Set myFolder = myNamespace.GetDefaultFolder(olFolderCalendar)

becomes:

Set myFolder = myNamespace.GetDefaultFolder(9)
GetPageInfo Method

Returns information about a custom property page.

`expression.GetPageInfo(HelpFile, HelpContext)`

`expression`  Required. An expression that returns a `PropertyPage` object.

**HelpFile**  Required `String`. Specifies the Help file associated with the property page.

**HelpContext**  Required `Long`. Specifies the context ID of the Help topic associated with the property page.
Example

This Microsoft Visual Basic/Visual Basic for Applications example returns the name of the Help file and the context ID of the topic to be displayed.

Private Sub PropertyPage_GetPageInfo(HelpFile As String, HelpContext
  HelpFile = "ProjPage.chm"
  HelpContext = IDH_PageInfo
End Sub
GetPrevious Method

The **GetPrevious** method returns the previous object in the specified collection. It returns **Nothing** if no previous object exists, for example, if already positioned at the beginning of the collection.

**Note** To ensure correct operation of the **GetFirst**, **GetLast**, **GetNext**, and **GetPrevious** methods in a large collection, call **GetFirst** before calling **GetNext** on that collection, and call **GetLast** before calling **GetPrevious**. To ensure that you are always making the calls on the same collection, create an explicit variable that refers to that collection before entering the loop.

**expression.GetPrevious**

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

Returns a **Recipient** object identified by the specified entry ID (if valid). This method is used for ease of transition between MAPI and OLE/Messaging applications and Microsoft Outlook.

*expression*.GetRecipientFromID(*EntryID*)

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

*EntryID*  Required **String**. The **EntryID** of the recipient.
GetRecurrencePattern Method

Returns a RecurrencePattern object that represents the recurrence attributes of an appointment or task. If there is no existing recurrence pattern, a new, empty RecurrencePattern object is returned.

expression.GetRecurrencePattern

expression Required. An expression that returns an AppointmentItem or TaskItem object.
Example

This Visual Basic for Applications example uses `GetRecurrencePattern` to obtain the `RecurrencePattern` for an appointment.

```vbnet
Set myOlApp = CreateObject("Outlook.Application")
Set myAppt = myOlApp.CreateItem(olAppointmentItem)
Set myPattern = myAppt.GetRecurrencePattern
```

If you use VBScript, you do not create the `Application` object, and you cannot use named constants. This example shows how to perform the same task using VBScript.

```vbnet
Set myAppt = Application.CreateItem(1)
Set myPattern = myAppt.GetRecurrencePattern
```
GetSharedDefaultFolder Method

Returns a MAPIFolder object that represents the specified default folder for the specified user. This method is most useful in the delegation scenario, where one user has delegated access to another user for one or more of their default folders (for example, their shared Calendar folder).

expression.GetSharedDefaultFolder(Recipient, FolderType)

expression  Required. An expression that returns a NameSpace object.

Recipient  Required Recipient object. The owner of the folder. The Recipient object must be resolved.

FolderType  Required OlDefaultFolders The type of folder.

OlDefaultFolders can be one of these OlDefaultFolders constants:

olFolderCalendar
olFolderContacts
olFolderDeletedItems
olFolderDrafts
olFolderInbox
olFolderJournal
olFolderNotes
olFolderOutbox
olFolderSentMail
olFolderSharedRoot
olFolderTasks
olPublicFoldersAllPublicFolders
Example

This Visual Basic for Applications example uses the `GetSharedDefaultFolder` method to resolve the `Recipient` object representing Kim Buhler, and then returns her shared default Calendar folder.

```vba
Set myOlApp = CreateObject("Outlook.Application")
Set myNameSpace = myOlApp.GetNamespace("MAPI")
Set myRecipient = myNameSpace.CreateRecipient("Kim Buhler")
myRecipient.Resolve
If myRecipient.Resolved Then
    Set KimCalendarFolder = _
        myNameSpace.GetSharedDefaultFolder(
            myRecipient, olFolderCalendar)
End If
```

If you use VBScript, you do not create the `Application` object, and you cannot use named constants. This example shows how to perform the same task using VBScript.

```vbs
Set myNameSpace = Application.GetNamespace("MAPI")
Set myRecipient = myNameSpace.CreateRecipient("Kim Buhler")
myRecipient.Resolve
If myRecipient.Resolved Then
    Set KimCalendarFolder = _
        myNameSpace.GetSharedDefaultFolder(
            myRecipient, 9)
End If
```
GoToDate Method

Changes the date information displayed in the current viewing window.

expression.GoToDate(Date)

expression  Required. An expression that returns a View object.

Date  Required Date. The date of the information you want to view.
**HideFormPage Method**

Hides a form page in the inspector.

\[ \text{expression}\cdot\text{HideFormPage}(\text{PageName}) \]

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

*PageName*  Required *String*. The display name of the page to be hidden.
Example

This Visual Basic for Applications example uses HideFormPage to hide the "General" page of a newly-created ContactItem and then displays the item.

```
Set myOlApp = CreateObject("Outlook.Application")
Set myItem = myOlApp.CreateItem(olContactItem)
Set myPages = myItem.GetInspector.ModifiedFormPages
Set myPage = myPages.Add("General")
Set myInspector = myOlApp.ActiveInspector
myInspector.HideFormPage("General")
myItem.Display
```

If you use VBScript, you do not create the Application object, and you cannot use named constants. This example shows how to perform the same task using VBScript.

```
Set myItem = Application.CreateItem(2)
Set myPages = myItem.GetInspector.ModifiedFormPages
Set myPage = myPages.Add("General")
Set myInspector = Application.ActiveInspector
myInspector.HideFormPage("General")
myItem.Display
```
IsPaneVisible Method

Determines whether a specific explorer pane is visible. For example, use this method to determine if the Folder List is currently visible in the explorer. Returns **True** if the specified pane is displayed in the explorer.

**Note** You can also use the **Visible** property of the **OutlookBarPane** object to determine whether the Outlook Bar is visible.

```
object.IsPaneVisible(Pane)
```

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

*Pane* Required **OlPane**.

*OlPane* can be one of these OlPane constants.

- **OlFolderList**
- **OlOutlookBar**
- **OlPreview**
Example

This Microsoft Visual Basic/Visual Basic for Applications sample uses the `IsPaneVisible` method to determine whether the Outlook Bar is visible and uses the `ShowPane` method to display it if it is not visible.

```vba
Dim myOlApp As New Outlook.Application
Set myOlExp = myOlApp.ActiveExplorer
If myOlExp.IsPaneVisible(olOutlookBar) = False Then
    myOlExp.ShowPane olOutlookBar, True
End If
```

If you use VBScript, you do not create the `Application` object, and you cannot use named constants. This example shows how to perform the same task using VBScript.

```vbs
Set myOlExp = Application.ActiveExplorer
If myOlExp.IsPaneVisible(1) = False Then
    myOlExp.ShowPane 1, True
End If
```
IsSearchSynchronous Method

Returns a Boolean indicating if a search will be synchronous or asynchronous.

\[ expression.IsSearchSynchronous(LookInFolders) \]

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

**LookInFolders** Required **String**. The path name of the folders that the search will search through.
Example

The following example uses a function to test whether or not a given path name will result in a synchronous search.

```vba
Function Synch(ByVal Foldername As String, ByRef App As Application) As Boolean
    Synch = App.IsSearchSynchronous("Inbox")
End Function
```
IsWordMail Method

Determines whether the mail message associated with an inspector is displayed in an Outlook Inspector or in Microsoft Word. Returns True if the mail message is displayed in Microsoft Word (that is, if Word Mail is in use). The OlEditorType constant will be olEditorWord.

description

expression.IsWordMail

description Required. An expression that returns an Inspector object.
Item Method

Returns an object from a collection. The following table shows the collections supported and the object type returned.

<table>
<thead>
<tr>
<th>Collection</th>
<th>Object Returned</th>
</tr>
</thead>
<tbody>
<tr>
<td>Actions</td>
<td>Action</td>
</tr>
<tr>
<td>AddressEntries</td>
<td>AddressEntry</td>
</tr>
<tr>
<td>AddressLists</td>
<td>AddressList</td>
</tr>
<tr>
<td>Attachments</td>
<td>Attachment</td>
</tr>
<tr>
<td>Exceptions</td>
<td>Exception</td>
</tr>
<tr>
<td>Explorers</td>
<td>Explorer</td>
</tr>
<tr>
<td>Folders</td>
<td>MAPIFolder</td>
</tr>
<tr>
<td>Inspectors</td>
<td>Inspector</td>
</tr>
<tr>
<td>Items</td>
<td>Outlook item</td>
</tr>
<tr>
<td>Links</td>
<td>Link</td>
</tr>
<tr>
<td>OutlookBarGroups</td>
<td>OutlookBarGroup</td>
</tr>
<tr>
<td>OutlookBarShortcuts</td>
<td>OutlookBarShortcut</td>
</tr>
<tr>
<td>Pages</td>
<td>Page</td>
</tr>
<tr>
<td>Panes</td>
<td>Pane</td>
</tr>
<tr>
<td>PropertyPages</td>
<td>PropertyPage</td>
</tr>
<tr>
<td>Selection</td>
<td>Outlook item</td>
</tr>
<tr>
<td>SyncObjects</td>
<td>SyncObject</td>
</tr>
<tr>
<td>Recipients</td>
<td>Recipient</td>
</tr>
<tr>
<td>UserProperties</td>
<td>UserProperty</td>
</tr>
</tbody>
</table>

All other Microsoft Outlook collections

A generic **Object** representing a single object in the specified collection

expression.Item(*Index*)

expression  Required. An expression that returns a valid collection object.
**Index**  Required **Variant**. Either the index number of the object, or a value used to match the default property of an object in the collection.
Example

This Visual Basic for Applications example returns the second \texttt{MailItem} object in the default Inbox folder. It assumes that at least two \texttt{MailItem} objects already exist.

\begin{verbatim}
Set myOlApp = CreateObject("Outlook.Application")
Set myNameSpace = myOlApp.GetNameSpace("MAPI")
Set myFolder = _
               myNameSpace.GetDefaultFolder(olFolderInbox)
Set mySecondItem = myFolder.Items.Item(2)
\end{verbatim}

This Visual Basic for Applications example returns the "Forward" action from the \texttt{Actions} collection.

\begin{verbatim}
Set myOlApp = CreateObject("Outlook.Application")
Set myItem = myOlApp.ActiveInspector.CurrentItem
Set myAction = myItem.Actions.Item("Forward")
\end{verbatim}

This example returns a \texttt{MAPIFolder} object from a \texttt{Folders} collection.

\begin{verbatim}
Set myOlApp = CreateObject("Outlook.Application")
Set myNameSpace = myOlApp.GetNameSpace("MAPI")
Set myFolders = myNameSpace.Folders
Set myFolder = myFolders.Item("Public Folder")
\end{verbatim}

This Visual Basic for Applications example also returns a \texttt{MAPIFolder} object from a \texttt{Folders} collection.

\begin{verbatim}
Set myOlApp = CreateObject("Outlook.Application")
Set myFolders = _
               myOlApp.ActiveExplorer.CurrentFolder.Folders
Set myFolder = myFolders.Item("Project X")
\end{verbatim}

This Visual Basic for Applications example creates a contact, returns its empty \texttt{Pages} collection, adds three custom pages and then returns the first custom page from the collection.

\begin{verbatim}
Set myOlApp = CreateObject("Outlook.Application")
Set myItem = myOlApp.CreateItem(olContactItem)
Set myPages = myItem.GetInspector.ModifiedFormPages
myPages.Add "One"
\end{verbatim}
myPages.Add "Two"
myPages.Add "Three"
Set myPage = myPages.Item("One")

This Visual Basic for Applications example creates a mail message, adds four `Recipient` objects, then returns the name of the third recipient from the newly-created `Recipients` collection.

Set myOlApp = CreateObject("Outlook.Application")
Set myItem = myOlApp.CreateItem(olMailItem)
Set myRecipients = myItem.Recipients
myRecipients.Add("Rich Andrews")
myRecipients.Add("Robin Hjellin")
myRecipients.Add("Meng Phua")
myRecipients.Add("Kim Yoshida")
msgbox myRecipients.Item(3).name
Logoff Method

- Logs the user off from the current MAPI session.

expression.Logoff

expression An expression that returns a NameSpace object.
Logon Method

Logs the user on to MAPI, obtaining a MAPI session.

expression.Logon(Profile, Password, ShowDialog, NewSession)

expression  An expression that returns a NameSpace object.

Profile  Optional Variant. The profile name, as a String, to use for the session.

Password  Optional Variant. The password (if any), as a String, associated with the profile.

ShowDialog  Optional Variant. True to display the MAPI logon dialog.

NewSession  Optional Variant. True to create a new session (doesn't use an existing session). Multiple sessions cannot be created in Outlook.
Example

This Visual Basic for Applications example uses the **Logon** method to log on to a new session, displaying the dialog box to verify the profile name and password.

```vba
Set myOlApp = CreateObject("Outlook.Application")
Set myNameSpace = myOlApp.GetNameSpace("MAPI")
myNameSpace.Logon "myProfile", "myPassword", True, True
```
MarkComplete Method

Marks the task as completed. Sets **PercentComplete** to "100%", **Complete** to **True**, and **DateCompleted** to the current date.

*expression*.MarkComplete

*expression* Required. An expression that returns a **TaskItem** object.
Show All
Move Method

- Moves an Outlook item to a new folder.

expression.Move(DestFldr)

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

DestFldr Required. An expression that returns a MAPIFolder object. The destination folder.
Example

This Visual Basic for Applications example uses the `GetDefaultFolder` to return the `MAPIFolder` object that represents the default folder. It then uses the `Find` and `FindNext` methods to find all messages sent by Mike Nash and uses the `Move` method to move all mail messages sent by Mike Nash from the default Inbox folder to the Personal Mail folder.

```vba
Set myOlApp = CreateObject("Outlook.Application")
Set myNameSpace = myOlApp.GetNameSpace("MAPI")
Set myInbox = myNameSpace.GetDefaultFolder(olFolderInbox)
Set myItems = myInbox.Items
Set myDestFolder = myInbox.Folders("Personal Mail")
Set myItem = myItems.Find("[SenderName] = 'Mike Nash'")
While TypeName(myItem) <> "Nothing"
    myItem.Move myDestFolder
    Set myItem = myItems.FindNext
Wend
```

If you use VBScript, you do not create the `Application` object, and you cannot use named constants. This example shows how to perform the same task using VBScript.

```vba
Set myNameSpace = Application.GetNameSpace("MAPI")
Set myInbox = myNameSpace.GetDefaultFolder(6)
Set myItems = myInbox.Items
Set myDestFolder = myInbox.Folders("Personal Mail")
Set myItem = myItems.Find("[SenderName] = 'Mike Nash'")
While TypeName(myItem) <> "Nothing"
    myItem.Move myDestFolder
    Set myItem = myItems.FindNext
Wend
```
MoveTo Method

Moves a folder to the specified destination folder.

expression.MoveTo(DestinationFolder)

expression  Required. An expression that returns a MAPIFolder object.

DestinationFolder  Required. An expression that returns a MAPIFolder object. The destination folder for the folder that is being moved.
This Visual Basic for Applications example adds a new folder to the default Contacts folder, and then uses the MoveTo method to move it to the default Inbox folder.

```vba
Set myOlApp = CreateObject("Outlook.Application")
Set myNameSpace = myOlApp.GetNamespace("MAPI")
Set myFolder = myNameSpace.GetDefaultFolder(olFolderContacts)
Set myNewFolder = myFolder.Folders.Add("My Contacts")
myNewFolder.MoveTo myNameSpace.GetDefaultFolder(olFolderInbox)
```

If you use VBScript, you do not create the Application object, and you cannot use named constants. This example shows how to perform the same task using VBScript.

```vbs
Set myNameSpace = Application.GetNamespace("MAPI")
Set myFolder = myNameSpace.GetDefaultFolder(10)
Set myNewFolder = myFolder.Folders.Add("My Contacts")
myNewFolder.MoveTo myNameSpace.GetDefaultFolder(6)
```
OnStatusChange Method

Notifies Microsoft Outlook that a custom property page has changed.

expression.OnStatusChange

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

This Microsoft Visual Basic/Visual Basic for Applications example shows how to call the `OnStatusChange` method to notify Microsoft Outlook that the user has changed a value on a custom property page.

Private Sub Option1_Click()
    Dim myPPSite As Outlook.PropertyPageSite
    Set myPPSite = Parent
    If Not TypeName(myPPSite) = "Nothing" Then
        globNewUserType = globAdministrator
        If globUserType <> globNewUserType Then
            globDirty = True
            myPPSite.OnStatusChange
        End If
    Else
        If TypeName(myPPSite) = "Nothing" Then
            MsgBox "The Property Page returned an empty result."
        End If
    End If
End Sub
PickFolder Method

The **PickFolder** method displays the Pick Folder dialog box. This is a modal dialog box which means that code execution will not continue until the user either selects a folder or cancels the dialog box. Returns a [MAPIFolder](#) object corresponding to the folder that the user selects in the dialog box. Returns **Nothing** when the dialog box is canceled by the user.

```expression.PickFolder```

*expression*  Required. A [NameSpace](#) object.
Post Method

Sends (posts) the **PostItem** object. The **Post** method, which is analogous to the **Send** method for the **MailItem** object, is used to save the post to the target public folder instead of mailing it.

`expression.Post`

**expression** Required. An expression that returns a **PostItem** object.
Show All
PrintOut Method

Prints the Outlook item using all default settings.

Note The PrintOut method is the only Outlook method that can be used for printing.

expression.PrintOut

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

Saves the definition of the FormDescription object in the specified form registry (library).

Forms are registered as one of three classes: Folder, Organization, or Personal. The Folder form registry holds a set of forms that are only accessible from that specific folder, whether public or private. The Organization form registry holds forms that are shared across an entire enterprise and thus are accessible to everyone. The Personal form registry holds forms that are accessible only to the current store user.

**Note** The Name property must be set before you can use the PublishForm method.

```expression.PublishForm(Registry, Folder)```

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

**Registry**  Required OlFormRegistry. The form class.

OlFormRegistry can be one of these OlFormRegistry constants.

- **olDefaultRegistry** Handles items without regard to their form class.
- **olFolderRegistry**
- **olOrganizationRegistry**
- **olPersonalRegistry**

**Folder**  Optional except with olFolderRegistry. Expression that returns a MAPIFolder object. Used only with Folder form registry. The folder object from which the forms must be accessed.
Example

This Visual Basic for Applications example creates a contact, obtains its **FormDescription** object, and saves it in the Folder form registry of the default Contacts folder.

**Note** The **PublishForm** method will return an error if the caption (**Name**) for the form is not set first.

```vba
Set myOlApp = CreateObject("Outlook.Application")
Set myNameSpace = myOlApp.GetNameSpace("MAPI")
Set myFolder = myNameSpace.GetDefaultFolder(olFolderContacts)
Set myItem = myOlApp.CreateItem(olContactItem)
Set myForm = myItem.FormDescription
myForm.Name = "My Contact"
myForm.PublishForm olFolderRegistry, myFolder
```

This Visual Basic for Applications example creates an appointment, obtains its **FormDescription** object, and saves it in the user's Personal form registry.

To view the form after you have published it, go to **Choose Form** in the **File**, **New** menu and choose **Personal Forms Library** in the **Look In**: box. You can double-click on your new form, "Interview Scheduler" in the list.

```vba
Set myOlApp = CreateObject("Outlook.Application")
Set myItem = myOlApp.CreateItem(olAppointmentItem)
Set myForm = myItem.FormDescription
myForm.Name = "Interview Scheduler"
myForm.PublishForm olPersonalRegistry
```
Quit Method

Closes all currently open windows. The associated Microsoft Outlook session is closed completely; the user is logged out of the messaging system and any changes to items not already saved are discarded.

expression.Quit

expression Required. An expression that returns an Application object.
Remove Method

- Remove method as it applies to the Actions, Attachments, Folders, Items, ItemProperties, Pages, Recipients, and UserProperties objects.

Removes an object from one of the above collections.

(expression).Remove(Index)

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

Index Required Long. The index value of the object within the collection.

- Remove method as it applies to the Links, OutlookBarGroups, OutlookBarShortcuts, PropertyPages, Reminders, and Views objects.

Removes an object from the specified list.

(expression).Remove(Index)

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

Index Required Variant. The name or ordinal value of an object within a list.
Example

- As it applies to the **Actions, Attachments, Folders, Items, ItemProperties, Pages, Recipients, and UserProperties** objects.

This Visual Basic for Applications example uses the **Remove** method to remove all attachments from a forwarded message before sending it on to Jeff Smith.

```vbnet
Set myOlApp = CreateObject("Outlook.Application")
Set myItem = myOlApp.ActiveInspector.CurrentItem.Forward
Set myAttachments = myItem.Attachments
While myAttachments.Count > 0
    myAttachments.Remove 1
Wend
myItem.Recipients.Add "Jeff Smith"
myItem.Send
```

If you use VBScript, you do not create the **Application** object. This example shows how to perform the same task using VBScript.

```vbnet
Set myItem = Application.ActiveInspector.CurrentItem.Forward
Set myAttachments = myItem.Attachments
While myAttachments.Count > 0
    myAttachments.Remove 1
Wend
myItem.Recipients.Add "Jeff Smith"
myItem.Send
```

- As it applies to the **Links, OutlookBarGroups, OutlookBarShortcuts, PropertyPages, Reminders, and Views** objects.

The following example removes a **View** object from the **Views** collection.

```vbnet
Sub DeleteView()
    'Deletes a view from the collection
    Dim olApp As Outlook.Application
    Dim objName As NameSpace
    Dim objViews As Views
    Dim objView As View
    Dim strName As String
```
strName = "New Icon View"
Set olApp = Outlook.Application
Set objName = olApp.GetNamespace("MAPI")
Set objViews = objName.GetDefaultFolder(olFolderNotes).Views

    For Each objView In objViews
        If objView.Name = strName Then
            objViews.Remove (strName)
        End If
    Next objView

End Sub
**RemoveMember Method**

Removes an individual member from a given distribution list.

*expression.RemoveMember(Recipient)*

*expression*  Required. An expression that returns a [DistListItem](#) object.

*Recipient*  Required *Recipient*. The Recipient to be removed from the distribution list.
Example

The following example removes a member from the distribution list called Group List. The **RemoveMember** method will fail if the specified recipient is not valid.

```vba
Sub RemoveRec()
' Remove a recipient from the list and displays new list.

    Dim olApp As Outlook.Application
    Dim objDstList As DistListItem
    Dim objName As NameSpace
    Dim objRcpnt As Recipient
    Dim objMail As MailItem

    Set olApp = Outlook.Application
    Set objName = olApp.GetNamespace("MAPI")
    Set objDstList = objName.GetDefaultFolder(olFolderContacts).Items("Group List")
    Set objMail = olApp.CreateItem(olMailItem)
    Set objRcpnt = objMail.Recipients.Add(Name:="Jeff Smith")
    objDstList.RemoveMember Recipient:=objRcpnt
    objDstList.Display
    objDstList.Body = "Last Modified: " & Now

End Sub
```
RemoveMembers Method

Removes members from a distribution list.

expression.RemoveMembers(Recipients)

expression Required. An expression that returns a DistListItem object.

Recipients Required Recipients. The members to be removed from the distribution list.
Example

This Microsoft Visual Basic/Visual Basic for Applications example locates every distribution list item in the default Contacts folder and then determines if the current user is a member of the list. If that is the case, the current user is removed from the distribution list.

```vba
Dim myOlApp As New Outlook.Application
Dim myNameSpace As Outlook.NameSpace
Dim myFolder As Outlook.MAPIFolder
Dim myDistList As Outlook.DistListItem
Dim tmpRecips As Outlook.Recipients
Set myNameSpace = myOlApp.GetNamespace("MAPI")
Set tmpRecips = myOlApp.CreateItem(olMailItem).Recipients
tmpRecips.Add myNameSpace.CurrentUser.Name
Set myFolder = myNameSpace.GetDefaultFolder(olFolderContacts)
For x = 1 To myFolder.Items.Count
    If TypeName(myFolder.Items.Item(x)) = "DistListItem" Then
        Set myDistList = myFolder.Items.Item(x)
        For y = 1 To myDistList.MemberCount
            If myDistList.GetMember(y).Name = myNameSpace.CurrentUser.Name Then
                myDistList.RemoveMembers tmpRecips
                myDistList.Save
                Exit For
            End If
        Next y
    End If
Next x
```

If you use VBScript, you do not create the `Application` object, and you cannot use named constants. This example shows how to perform the same task using VBScript.

```vbs
Set myNameSpace = Application.GetNamespace("MAPI")
Set tmpRecips = Application.CreateItem(olMailItem).Recipients
tmpRecips.Add myNameSpace.CurrentUser.Name
Set myFolder = myNameSpace.GetDefaultFolder(10)
For x = 1 To myFolder.Items.Count
    If TypeName(myFolder.Items.Item(x)) = "DistListItem" Then
        Set myDistList = myFolder.Items.Item(x)
```
For y = 1 To myDistList.MemberCount
    If myDistList.GetMember(y).Name = _
        myNameSpace.CurrentUser.Name Then
        myDistList.RemoveMembers tmpRecips
        myDistList.Save
        Exit For
    End If
Next
End If
Next
RemoveStore Method

Removes a folder from the Folder List in the user interface.

`expression.RemoveStore(Folder)`

- `expression` Required. An expression that returns a `_namespace` object.
- `Folder` Required `MAPIFolder` object. The folder to be deleted from the list.
Remarks

You cannot remove a store from the main mailbox on the server.
**Example**

The following subroutine removes a folder called Personal Folders from the list of folders.

Sub RemovePST()
    Dim objName As NameSpace
    Dim objFolder As MAPIFolder

    Set objName = Application.GetNamespace("MAPI")
    Set objFolder = objName.Folders.Item("Personal Folders")
    objName.RemoveStore objFolder
End Sub
**Reply Method**

Creates a reply, pre-addressed to the original sender, from the original message. Returns the reply as a `MailItem` object.

`expression.Reply`  

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

Creates a reply to all original recipients from the original message. Returns the reply as a MailItem object.

expression.ReplyAll

definition

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

- Resets a built-in Microsoft Outlook view to its original settings.

`expression.Reset`

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

This method works only on built-in Outlook views.
Example

The following example resets all built-in views in the user's Inbox to their original settings. The **Standard** property is returned to determine if the view is a built-in Outlook view.

```vba
Sub ResetViews()
' Resets all standard views in the user's Inbox

    Dim olApp As Outlook.Application
    Dim objName As NameSpace
    Dim objViews As Views
    Dim objView As View

    Set olApp = Outlook.Application
    Set objName = olApp.GetNamespace("MAPI")
    Set objViews = objName.GetDefaultFolder(olFolderInbox).Views
    For Each objView In objViews
        If objView.Standard = True Then
            objView.Reset
        End If
    Next objView

End Sub
```
The **ResetColumns** method clears the properties that have been cached with the **SetColumns** method. All properties are accessible after calling the **ResetColumns** method. **SetColumns** should be reused to store new properties again. **ResetColumns** does nothing if **SetColumns** has not been called first.

**expression**.**ResetColumns**

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

Attempts to resolve a Recipient object against the Address Book. Returns True if the object was resolved, False if it was not.

expression.Resolve

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

This Visual Basic for Applications example uses the **Resolve** method to attempt to resolve a recipient and, if unsuccessful, displays the **item** so a user can resolve it manually.

```vbs
Set myOlApp = CreateObject("Outlook.Application")
Set myItem = myOlApp.CreateItem(olMailItem)
Set myRecipient = myItem.Recipients.Add("Tim O'Brien")
If Not myRecipient.Resolve Then myItem.Display
```

If you use VBScript, you do not create the **Application** object, and you cannot use named constants. This example shows how to perform the same task using VBScript.

```vbs
Set myItem = Application.CreateItem(0)
Set myRecipient = myItem.Recipients.Add("Tim O'Brien")
If Not myRecipient.Resolve Then myItem.Display
```
ResolveAll Method

Attempts to resolve all the `Recipient` objects in the `Recipients` collection against the Address Book. Returns `True` if all of the objects were resolved, `False` if one or more were not.

`expression.ResolveAll`

`expression` Required. An expression that returns a `Recipients` collection.
Example

This Visual Basic for Applications example uses the `ResolveAll` method to attempt to resolve all recipients and, if unsuccessful, displays a message box for each unresolved recipient.

```vba
Set myOlApp = CreateObject("Outlook.Application")
Set myItem = myOlApp.CreateItem(olMailItem)
Set myRecipients = myItem.Recipients
myRecipients.Add("Aaron Con")
myRecipients.Add("Viki Parrott")
myRecipients.Add("Jeffrey Weems")
If Not myRecipients.ResolveAll Then
    For Each myRecipient In myRecipients
        If Not myRecipient.Resolved Then
            MsgBox myRecipient.Name
        End If
    Next
End If
```

If you use VBScript, you do not create the `Application` object, and you cannot use named constants. This example shows how to perform the same task using VBScript.

```vbs
Set myItem = Application.CreateItem(0)
Set myRecipients = myItem.Recipients
myRecipients.Add("Aaron Con")
myRecipients.Add("Viki Parrott")
myRecipients.Add("Jeffrey Weems")
If Not myRecipients.ResolveAll Then
    For Each myRecipient In myRecipients
        If Not myRecipient.Resolved Then
            MsgBox myRecipient.Name
        End If
    Next
End If
```
Respond Method

Responds to a meeting request for the **AppointmentItem** object or a task request for the **TaskItem** object.

`expression.Respond(Response, fNoUI, fAdditionalTextDialog)`

- **expression** Required. An expression that returns an **AppointmentItem** or **TaskItem** object.

- **Response** Required  **OlMeetingResponse**. The response to the request.

  - OlMeetingResponse can be one of these OlMeetingResponse constants. For an **AppointmentItem** object:
    - olMeetingAccepted
    - olMeetingDeclined
    - olMeetingTentative
  - For a **TaskItem** object:
    - olTaskAccept
    - olTaskAssign
    - olTaskDecline
    - olTaskSimple

- **fNoUI** Optional for **AppointmentItem**, required for **TaskItem**. Boolean. **True** to not display a dialog box; the response is sent automatically. **False** to display the dialog box for responding.

- **fAdditionalTextDialog** Optional for **AppointmentItem**, required for **TaskItem**. Boolean. **False** to not prompt the user for input; the response is displayed in the inspector for editing. **True** to prompt the user to either send or send with comments. This argument is valid only if **fNoUI** is **False**.

**Note** The possible values for the optional parameters, **fNoUI** and
fAdditionalTextDialog and the subsequent results are as follows:

<table>
<thead>
<tr>
<th>fNoUI, fAdditionalTextDialog</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>True, True</td>
<td>Response item is returned with no user interface. To send the response, you must call the Send method.</td>
</tr>
<tr>
<td></td>
<td>For AppointmentItem and TaskItem:</td>
</tr>
<tr>
<td>True, False</td>
<td>Same result as with True, True.</td>
</tr>
<tr>
<td></td>
<td>For AppointmentItem:</td>
</tr>
<tr>
<td></td>
<td>Prompts user to Send or Edit before sending response.</td>
</tr>
<tr>
<td>False, True</td>
<td>For TaskItem:</td>
</tr>
<tr>
<td></td>
<td>If Display method has been called, the user prompt appears. Otherwise, the item is sent without prompting and resulting item is nothing.</td>
</tr>
<tr>
<td>False, False</td>
<td>For AppointmentItem:</td>
</tr>
<tr>
<td></td>
<td>New response item appears in the user interface, but no prompt.</td>
</tr>
<tr>
<td>False, False</td>
<td>For TaskItem:</td>
</tr>
<tr>
<td></td>
<td>Does nothing.</td>
</tr>
</tbody>
</table>
Example

This Visual Basic for Applications example uses the **Find** method to locate a **MeetingItem** in the user's Inbox. If there are no meeting items, the user is informed. If a meeting item is located, the **GetAssociatedAppointment** method is called to get the **AppointmentItem** and the **Respond** method allows acceptance.

```vba
Set myOlApp = CreateObject("Outlook.Application")
Set myNamespace = myOlApp.GetNamespace("MAPI")
Set myFolder = myNamespace.GetDefaultFolder(olFolderInbox)
Set myMtgReq = myFolder.Items.Find("[MessageClass] = 'IPM.Schedule.Meeting.Request'"
If TypeName(myMtgReq) <> "Nothing" Then
  Set myAppt = myMtgReq.GetAssociatedAppointment(True)
  myAppt.Respond olResponseAccepted, False, True
Else
  MsgBox "You have no meeting requests."
End If
```

If you use VBScript, you do not create the **Application** object, and you cannot use named constants. This example shows how to perform the same task using VBScript.

```vba
Set myNamespace = Application.GetNamespace("MAPI")
Set myFolder = myNamespace.GetDefaultFolder(6)
Set myMtgReq = myFolder.Items.Find("[MessageClass] = 'IPM.Schedule.Meeting.Request'"
If TypeName(myMtgReq) <> "Nothing" Then
  Set myAppt = myMtgReq.GetAssociatedAppointment(True)
  myAppt.Respond olResponseAccepted, False, True
Else
  MsgBox "You have no meeting requests."
End If
```
Restrict Method

Applies a filter to the Items collection, returning a new collection containing all items from the original which match the filter. This method is an alternative to using the Find method or FindNext method to iterate over specific items within a collection. The Find or FindNext methods are faster than filtering if there are a small number of items. The Restrict method is significantly faster if there is a large number of items in the collection, especially if only a few items in the large collection are expected to be found.

**Note** If you are using user-defined fields as part of a Find or Restrict clause, the user-defined fields must exist in the folder, otherwise the code will generate an error stating that the field is unknown. You can add a field to a folder by displaying the Field Chooser and clicking New.

\[
expression.\text{Restrict}(\text{Filter})
\]

- **expression** Required. An expression that returns an Items object.
- **Filter** Required String. A filter string expression to be applied. For details, see the Find method.
Remarks

This method cannot be used, and will cause an error, with the following properties:

Body
Categories
Children
Class
Companies
CompanyLastFirstNoSpace
CompanyLastFirstSpaceOnly
ContactNames
Contacts
ConversationIndex
DLName
Email1EntryID
Email2EntryID
Email3EntryID
EntryID
HTMLBody
IsOnlineMeeting

LastFirstNoSpaceCompany
LastFirstSpaceOnly
LastFirstSpaceOnlyCompany
MemberCount
NetMeetingAlias
NetMeetingAutoStart
NetMeetingOrganizerAlias
NetMeetingServer
NetMeetingType
RecurrenceState
ReplyRecipients
ReceivedByEntryID
ReceivedOnBehalfOfEntryID
ResponseState
Saved
Sent
Submitted
LastFirstAndSuffix	VotingOptions
LastFirstNoSpace
Example

This Visual Basic for Applications example uses the **Restrict** method to get all Inbox items dealing with Project X and moves them to the Project X folder.

```vba
Set myOlApp = CreateObject("Outlook.Application")
Set myNameSpace = myOlApp.GetNamespace("MAPI")
Set myFolder = myNameSpace.GetDefaultFolder(olFolderInbox)
Set myItems = myFolder.Items
Set myRestrictItems = myItems.Restrict "[Categories] = 'Project X'"
For Each myItem In myRestrictItems
    myItem.Move myFolder.Folders("Project X")
Next
```

If you use VBScript, you do not create the **Application** object, and you cannot use named constants. This example shows how to perform the same task using VBScript.

```vba
Set myNameSpace = Application.GetNamespace("MAPI")
Set myFolder = myNameSpace.GetDefaultFolder(6)
Set myItems = myFolder.Items
Set myRestrictItems = myItems.Restrict "[Categories] = 'Project X'"
For Each myItem In myRestrictItems
    myItem.Move myFolder.Folders("Project X")
Next
```

This Visual Basic for Applications example uses the **Restrict** method to apply a filter to the Contact items based on the item's **LastModificationTime** property.

```vba
Public Sub ContactDateCheck()
    Set myOlApp = CreateObject("Outlook.Application")
    Set myNameSpace = myOlApp.GetNamespace("MAPI")
    Set myContacts = myNameSpace.GetDefaultFolder(olFolderContacts).Items
    Set myItems = myContacts.Restrict("[LastModificationTime] > '05/1"
    For Each myItem In myItems
        MsgBox myItem.FullName & "": " & myItem.LastModificationTime
    Next
End Sub
```

The following Visual Basic for Applications example is the same as the example above, except that it demonstrates the use of a variable in the filter.
Public Sub ContactDateCheck()
    Set myOlApp = CreateObject("Outlook.Application")
    Set myNameSpace = myOlApp.GetNamespace("MAPI")
    Set myContacts = myNameSpace.GetDefaultFolder(olFolderContacts).Items
    DateStart = #6/11/97#
    DateToCheck$ = "[LastModificationTime] >= """ & DateStart & """
    Set myRestrictItems = myContacts.Restrict(DateToCheck$)
    For Each myItem In myRestrictItems
        MsgBox myItem.FullName & ": " & myItem.LastModificationTime
    Next
End Sub
Show All
Save Method

Saves the Outlook item to the current folder or, if this is a new item, to the Outlook default folder for the item type.

expression.Save

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

This Visual Basic for Applications example creates a task called "Oil Change" that recurs every three months and uses the `Regenerate` property to set it to regenerate after each recurrence.

Set myOlApp = CreateObject("Outlook.Application")
Set myItem = myOlApp.CreateItem(olTaskItem)
Set myPattern = myItem.GetRecurrencePattern
myPattern.RecurrenceType = olRecursMonthly
myPattern.Regenerate = True
myPattern.Interval = 3
myItem.Subject = "Oil Change"
myItem.Save
myItem.Display

If you use VBScript, you do not create the `Application` object, and you cannot use named constants. This example shows how to perform the same task using VBScript.

Set myItem = Application.CreateItem(3)
Set myPattern = myItem.GetRecurrencePattern
myPattern.RecurrenceType = 2
myPattern.Regenerate = True
myPattern.Interval = 3
myItem.Subject = "Oil Change"
myItem.Save
myItem.Display
Show All
SaveAs Method

Saves the Outlook item to the specified path and in the format of the specified file type. If the file type is not specified, the MSG format is used.

`expression.SaveAs(Path, Type)`

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

**Path** Required **String**. The path in which to save the item.

**Type** Optional **Variant**. The file type to save. Can be one of the following OlSaveAsType constants: `olDoc`, `olHTML`, `olMSG`, `olRTF`, `olTemplate`, `olTXT`, `olVCal`, or `olVCard`. 
Example

This Visual Basic for Applications example uses the **SaveAs** method to save the currently open **item** as a text file in the My Documents folder, using the subject as the file name.

```
Dim myItem As Inspector
Dim objItem As Object
Set myOlApp = CreateObject("Outlook.Application")
Set myItem = myOlApp.ActiveInspector
If Not TypeName(myItem) = "Nothing" Then
    Set objItem = myItem.CurrentItem
    strname = objItem.Subject
    objItem.SaveAs "C:\" & " & strname & .txt", olTXT
Else
    MsgBox "There is no current active Inspector."
End If
```

If you use VBScript, you do not create the **Application** object, and you cannot use named constants. This example shows how to perform the same task using VBScript.

```
Set myItem = Application.ActiveInspector
If Not TypeName(myItem) = "Nothing" Then
    Set objItem = myItem.CurrentItem
    strname = objItem.Subject
    objItem.SaveAs "C:\" & " & strname & .txt", 0
Else
    MsgBox "There is no current active Inspector."
End If
```
SaveAsFile Method

Saves the attachment to the specified path.

expression.SaveAsFile(Path)

expression  Required. An expression that returns an Attachment object.

Path  Required String. The location at which to save the attachment.
Example

This Visual Basic for Applications example uses the `SaveAsFile` method to save the first attachment of the currently open item as a file in the My Documents folder, using the attachment's display name as the file name.

```vba
Set myOlApp = CreateObject("Outlook.Application")
Set myItem = myOlApp.ActiveInspector.CurrentItem
Set myAttachments = myItem.Attachments
myAttachments.Item(1).SaveAsFile "C:\My Documents" & _
    myAttachments.Item(1).DisplayName
```

If you use VBScript, you do not create the `Application` object. This example shows how to perform the same task using VBScript.

```vbs
Set myItem = Application.ActiveInspector.CurrentItem
Set myAttachments = myItem.Attachments
myAttachments.Item(1).SaveAsFile "C:\My Documents" & _
    myAttachments.Item(1).DisplayName
```
Send Method

 Sends the appointment, meeting item, mail message, or task.

expression.Send

expression Required. An expression that returns an AppointmentItem, MeetingItem, MailItem, or TaskItem object.

Example

This Visual Basic for Applications example uses the GetDefaultFolder method to return the MAPIFolder object that represents the default Inbox folder for the current user. It then uses the Forward method to retrieve the first message in the default Inbox folder and forward it to Laura Jennings. The Add method is used to add Laura Jennings to the Recipients collection and the Send method sends the item to all recipients. It is assumed that the name will resolve unambiguously in the Address Book.

Set myOlApp = CreateObject("Outlook.Application")
Set myNameSpace = myOlApp.GetNameSpace("MAPI")
Set myFolder = myNameSpace.GetDefaultFolder(olFolderInbox)
Set myForward = myFolder.Items(1).Forward
myForward.Recipients.Add "Laura Jennings"
myForward.Send

If you use VBScript, you do not create the Application object, and you cannot use named constants. This example shows how to perform the same task using VBScript.

Set myNameSpace = Application.GetNameSpace("MAPI")
Set myFolder = myNameSpace.GetDefaultFolder(6)
Set myForward = myFolder.Items(1).Forward
myForward.Recipients.Add "Laura Jennings"
myForward.Send
**SetColumns Method**

The `SetColumns` method allows the user to cache certain properties for extremely fast access to those particular properties of the item. The `SetColumns` method is useful for iterating through the `Items` object. If you don't use this method, Microsoft Outlook must open each item to access the property. With the `SetColumns` method, Outlook only checks the properties that you have cached. Properties which are not cached are returned empty.

```vba
expression.SetColumns(Columns)
```

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

- **Columns** Required. A `String` containing the names of the properties to be cached, separated by commas.

**Remarks**

For the `Items` object, `SetColumns` cannot be used, and will cause an error, with any property that returns an object, and it cannot be used with the following properties:

**Body**

- **Categories**
- **Children**
- **Class**
- **Companies**
- **Contacts**
- **DLName**

- **MemberCount**
- **RecurrenceState**
- **ReplyRecipients**
- **ResponseState**
- **Saved**
- **Sent**
<table>
<thead>
<tr>
<th>EntryID</th>
<th>Submitted</th>
</tr>
</thead>
<tbody>
<tr>
<td>HTMLBody</td>
<td>VotingOptions</td>
</tr>
</tbody>
</table>
SetCurrentFormPage Method

Displays the specified form page in the inspector.

`expression.SetCurrentFormPage(PageName)`

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

*PageName* Required **String**. The display name of the form page.
Example

This VBScript example uses the Open event to display the "All Fields" page every time the item is opened.

Function Item_Open()
    Item.GetInspector.SetCurrentFormPage "All Fields"
End Function
SetIcon Method

Sets the icon for the specified Microsoft Outlook bar shortcut.

expression.SetIcon(Icon)

expression  Required. An expression that returns an OutlookBarShortcut object.

Icon  Required Variant. The path of the icon.
Example

The following example sets the icon of all shortcuts with the target **MAPIFolder** object to the icon image Mail.ico located on the user's computer. The example assumes that this icon exists in the specified location.

Sub SetFolderIcon()
    Dim OlApp As New Outlook.Application
    Dim objOlBar As Outlook.OutlookBarPane
    Dim objolGroup As Outlook.OutlookBarGroup
    Dim objOlShortcuts As Outlook.OutlookBarShortcuts
    Dim objOlShortcut As Outlook.OutlookBarShortcut
    Dim intTop As Integer
    Dim i As Integer
    Set objOlBar = OlApp.ActiveExplorer.Panes.Item("OutlookBar")
    Set objolGroup = objOlBar.Contents.Groups.Item(1)
    Set objOlShortcuts = objolGroup.Shortcuts
    intTop = objOlShortcuts.Count
    For i = intTop To 1 Step -1
        Set objOlShortcut = objOlShortcuts.Item(i)
        If TypeName(objOlShortcut.Target) = "MAPIFolder" Then
            objOlShortcut.SetIcon _
                ("C:\Program Files\Microsoft Office\Office\forms\1033\Mail.ico")
        End If
    Next i
End Sub
ShowCategoriesDialog Method

Displays the Show Categories dialog box, which allows you to select categories that correspond to the subject of the item.

expression.ShowCategoriesDialog

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

The following example creates a new appointment item, displays the item on the screen, and opens up the Show Categories dialog box.

Sub Appointment()
'Creates an appointment item to access Fields property
    Dim appolApp As Outlook.Application
    Dim olApptItem As AppointmentItem
    'Create an instance of the application
    Set appolApp = Outlook.Application
    'Create appointment item
    Set olApptItem = appolApp.CreateItem(olAppointmentItem)

    olApptItem.Body = "Please meet with me regarding these sales fig
    olApptItem.Recipients.Add ("Jeff Smith")
    olApptItem.Subject = "Low Sales Reports"
    'Display the item
    olApptItem.Display
    'Display the Show categories dialog
    olApptItem.ShowCategoriesDialog

End Sub
ShowFormPage Method

Shows a form page in the inspector.

`expression.ShowFormPage(PageName)`

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

*PageName* Required *String*. The display name of the page to be shown.
Example

This Visual Basic for Applications example uses the `ShowFormPage` method to show the All Fields page of the currently open item. If there is no currently open item, a message box will inform the user.

```vba
Set myOlApp = CreateObject("Outlook.Application")
Set myInspector = myOlApp.ActiveInspector
On Error GoTo ErrorHandler
myInspector.ShowFormPage("All Fields")
Set myItem = myInspector.CurrentItem
myItem.Display
Exit Sub
ErrorHandler:
    MsgBox "No current item to display!"
```
Show All
ShowPane Method

Displays or hides a specific pane in the explorer.

**Note**  You can also use the **Visible** property of the **OutlookBarPane** object to display or hide the **Outlook Bar**.

*expression*.ShowPane(*Pane*, *Visible*)

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

*Pane*  Required **OlPane**.

OlPane can be one of these OlPane constants.

- **olFolderList**
- **olOutlookBar**
- **olPreview**

*Visible*  Required. **True** to make the pane visible, **False** to hide the pane.
Example

This Microsoft Visual Basic/Visual Basic for Applications example uses the ShowPane and IsPaneVisible methods to hide the preview pane if it is visible, or to display it if it is hidden.

Dim myOlApp As New Outlook.Application
Dim myOlExp As Outlook.Explorer
Set myOlExp = myOlApp.ActiveExplorer
myOlExp.ShowPane olPreview, _
    Not myOlExp.IsPaneVisible(olPreview)

If you use VBScript, you do not create the Application object. This example shows how to perform the same task using VBScript.

Set myOlExp = Application.ActiveExplorer
myOlExp.ShowPane 3, Not myOlExp.IsPaneVisible(3)
SkipRecurrence Method

Clears the current instance of a recurring task and sets the recurrence to the next instance of that task.

```
expression.SkipRecurrence
```

*expression* Required. An expression that returns a *TaskItem* object.
Snooze Method

Delays the reminder by a specified time. This is equivalent to the user clicking the Snooze button.

expression.Snooze(SnoozeTime)

expression Required. An expression that returns a Reminder object.

SnoozeTime Optional Variant. Indicates the amount of time (in minutes) to delay the reminder. The default value is 5 minutes.
Remarks

This method will fail if the current reminder is not active.
Example

The following example delays all active reminders by a specified amount of time.

Sub SnoozeReminders()
' Delays all reminders by a specified amount of time

    Dim olApp As Outlook.Application
    Dim objRems As Reminders
    Dim objRem As Reminder
    Dim varTime As Variant

    Set olApp = Outlook.Application
    Set objRems = olApp.Reminders
    varTime = InputBox("Enter the number of minutes to delay")

    For Each objRem In objRems
        If objRem.IsVisible = True Then
            objRem.Snooze (varTime)
        End If
    Next objRem

End Sub
### Sort Method

Sorts the collection of items by the specified property. The index for the collection is reset to 1 upon completion of this method.

**expression.Sort(Property, Descending, Order)**

- **expression** Required. An expression that returns one of the objects in the Applies To list.
- **Property** Required **String**. The name of the property by which to sort, which may be enclosed in brackets (for example, "[CompanyName]"). May not be a user-defined field, and may not be a multi-valued property, such as a category.
- **Descending** Applies to all objects in the Applies To list except the **AddressEntries** object. Optional **Variant** for the **Results** object; optional **Boolean** for all other objects. **True** to sort in descending order. The default value is **False** (ascending).
- **Order** Applies to the **AddressEntries** object only. Optional **Variant**. The order for the specified address entries. Can be one of these **OlSortOrder** constants: **olAscending**, **olDescending**, or **olSortNone**.

### Remarks

For the **Items** collection, **Sort** cannot be used, and will cause an error, with the following properties:

<table>
<thead>
<tr>
<th>Categories</th>
<th>LastFirstSpaceOnly</th>
</tr>
</thead>
<tbody>
<tr>
<td>Children</td>
<td>LastFirstSpaceOnlyCompany</td>
</tr>
<tr>
<td>Class</td>
<td>MemberCount</td>
</tr>
<tr>
<td>Companies</td>
<td>NetMeetingAlias</td>
</tr>
<tr>
<td>CompanyLastFirstNoSpace</td>
<td>NetMeetingAutoStart</td>
</tr>
<tr>
<td>-------------------------</td>
<td>---------------------</td>
</tr>
<tr>
<td>CompanyLastFirstSpaceOnly</td>
<td>NetMeetingOrganizerAlias</td>
</tr>
<tr>
<td>Contacts</td>
<td>NetMeetingServer</td>
</tr>
<tr>
<td>DLName</td>
<td>NetMeetingType</td>
</tr>
<tr>
<td>IsOnlineMeeting</td>
<td>RecurrenceState</td>
</tr>
<tr>
<td>LastFirstAndSuffix</td>
<td>ResponseState</td>
</tr>
<tr>
<td>LastFirstNoSpace</td>
<td>Sent</td>
</tr>
<tr>
<td>LastFirstNoSpaceCompany</td>
<td>Saved</td>
</tr>
</tbody>
</table>

**Sort** only affects the order of items in a collection. It does not affect the order of items in an explorer view.
Example

The following Visual Basic for Applications example uses the `Sort` method to sort the `Items` collection for the default Contacts folder by the "CompanyName" property and then displays the company names each in turn.

```
Set myOlApp = CreateObject("Outlook.Application")
Set myNamespace = myOlApp.GetNamespace("MAPI")
Set myFolder = _
    myNamespace.GetDefaultFolder(olFolderContacts)
Set myItems = myFolder.Items
myItems.Sort "[CompanyName]", False
For Each myItem in myItems
    MsgBox myItem.CompanyName
Next myItem
```

If you use VBScript, you do not create the `Application` object, and you cannot use named constants. This example shows how to perform the same task using VBScript.

```
Set myNamespace = Application.GetNamespace("MAPI")
Set myFolder = _
    myNamespace.GetDefaultFolder(10)
Set myItems = myFolder.Items
myItems.Sort "[CompanyName]", False
For Each myItem in myItems
    MsgBox myItem.CompanyName
Next myItem
```
Start Method

- Begins synchronizing a user’s folders using the specified synchronization profile.

expression.Start

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

This Microsoft Visual Basic/Visual Basic for Applications example starts synchronization of the user’s second synchronization profile in response to the user clicking a button on a form. The mySyncObj variable is declared as a public variable so it can be referenced by other modules.

Public mySyncObj As Outlook.SyncObject

Private Sub Command1_Click()
    Dim myOlApp As New Outlook.Application
    Dim mySyncs As Outlook.SyncObjects
    Set mySyncs = myOlApp.GetNamespace("MAPI").SyncObjects
    Set mySyncObj = mySyncs.Item(1)
    mySyncObj.Start
End Sub
StartTimer Method

Starts the timer on the journal entry. This method allows programmatic control of the timer function. The Duration, End and Start properties are automatically updated when the timer is stopped.

expression.StartTimer

expression  Required. An expression that returns a JournalItem object.
StatusReport Method

Sends a status report to all CC recipients (recipients returned by the StatusUpdateRecipients property) with the current status for the task. Returns an Object representing the status report.

expression.StatusReport

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

This Visual Basic for Applications example uses the `StatusReport` method to report the status of the currently open task.

```vbnet
Set myOlApp = CreateObject("Outlook.Application")
Set myinspector = myOlApp.ActiveInspector
If Not TypeName(myinspector) = "Nothing" Then
    If TypeName(myinspector.CurrentItem) = "TaskItem" Then
        Set myTask = myinspector.CurrentItem
        Set myReport = myTask.StatusReport
        myReport.Send
    End If
MsgBox "There is no task item currently open"
End If
MsgBox "There is no Inspector currently open"

If you use VBScript, you do not create the `Application` object. This example shows how to perform the same task using VBScript.

```vbnet
Set myinspector = Application.ActiveInspector
If Not TypeName(myinspector) = "Nothing" Then
    If TypeName(myinspector.CurrentItem) = "TaskItem" Then
        Set myTask = myinspector.CurrentItem
        Set myReport = myTask.StatusReport
        myReport.Send
    End If
MsgBox "There is no task item currently open"
End If
MsgBox "There is no Inspector currently open"
```
Stop Method

Immediately ends synchronizing a user’s folders using the specified synchronization profile. This method does not undo any synchronization that has already occurred.

expression.End

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

This Microsoft Visual Basic/Visual Basic for Applications example interrupts synchronization of the user’s second synchronization profile in response to the user clicking a button on a form. The mySyncObj variable is declared as a public variable so it can be referenced by other modules.

Public mySyncObj As Outlook.SyncObject

Private Sub Command2_Click()
    MsgBox "Synchronization stopped by user."
    mySyncObj.Stop
End Sub
StopTimer Method

Stops the timer on the journal entry. This method allows programmatic control of the timer function. The **Duration**, **End** and **Start** properties are automatically updated when the timer is stopped.

`expression.StopTimer`

*expression*  Required. An expression that returns a **JournalItem** object.
Update Method

The **Update** method posts a change to the **AddressEntry** object in the messaging system.

**expression.** `Update(MakePermanent, Refresh)`

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

**MakePermanent**  Optional **Variant**. A value of **True** indicates that the property cache is flushed and all changes are committed in the underlying address book. A value of **False** indicates that the property cache is flushed but not committed to persistent storage. The default value is **True**.

**Refresh**  Optional **Variant**. A value of **True** indicates that the property cache is reloaded from the values in the underlying address book. A value of **False** indicates that the property cache is not reloaded. The default value is **False**.
Remarks

New entries or changes to existing entries are not persisted in the collection until the Update method has been called with its MakePermanent parameter set to True.

To flush the cache and then reload the values from the address book, call Update with the MakePermanent parameter set to False and the Refresh parameter set to True.
Example

This Visual Basic for Applications example uses the **Add** method to add a new **AddressEntry** object to the Personal Address Book. The **Name** value, John Q. Public, is necessary to avoid errors.

If this entry already exists, the code skips to **DialogBox**. The **Details** method displays a dialog box that shows the information for this entry.

```vba
Set myOlApp = CreateObject("Outlook.Application")
Set myNamespace = myOlApp.GetNamespace("MAPI")
Set myAddrList = myNamespace.AddressLists("Personal Address Book")
Set myAddrEntries = myAddrList.AddressEntries
Set myEntry = myAddrEntries.Add("Microsoft Mail Address")
myEntry.Name = "John Q. Public"
On Error GoTo DialogBox
myEntry.Address = "someone@microsoft.com"
myEntry.Update
DialogBox:
myEntry.Details
```
Account Property

Returns or sets a **String** representing the account for the contact. Read/write.

*expression*.Account

*expression*  Required. An expression that returns a **ContactItem** object.
Show All
**Actions Property**

Returns an `Actions` collection that represents all the available actions for the Outlook item.

`expression.Actions`

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

This Visual Basic for Applications example creates a new mail message and uses the Add method to add an Action to it.

Set myOlApp = CreateObject("Outlook.Application")
Set myItem = myOlApp.CreateItem(olMailItem)
Set myAction = myItem.**Actions**.Add

If you use VBScript, you do not create the Application object, and you cannot use named constants. This example shows how to perform the same task using VBScript.

Set myItem = Application.CreateItem(0)
Set myAction = myItem.**Actions**.Add
ActualWork Property

Returns or sets a `Long` indicating the actual effort (in minutes) spent on the task. Read/write.

`expression.ActualWork`

`expression` Required. An expression that returns a `TaskItem` object.
Address Property

Returns or sets a String representing the e-mail address of the recipient. Read/write for the AddressEntry object; read-only for the Recipient object.

Note The Address property must be set before calling the Details method.

expression.Address

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

Returns or sets a **String** that indicates the Address Book name for the current **MAPIFolder** object. Read/write.

`expression.AddressBookName`

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

The following example changes the Address Book name for a given folder and displays the new name to the user. The subroutine accepts the folder object and a String representing the new folder name.

Sub BookName()
    Dim olApp As Outlook.Application
    Dim nmsName As NameSpace
    Dim fldFolder As MAPIFolder
    Dim strAns As String

    Set olApp = Outlook.Application
    'Create a reference to namespace
    Set nmsName = olApp.GetNamespace("MAPI")
    'Create an instance of the Inbox folder
    Set fldFolder = nmsName.GetDefaultFolder(olFolderInbox)
    'Prompt user for input
    strAns = InputBox("Enter the name of the new address book")
    'Call Sub procedure
    Call Changebook(fldFolder, strAns)
End Sub

Sub Changebook(ByVal fldFolder As MAPIFolder, ByVal strName As String)
    'Changes the name of the address book for a given folder
    fldFolder.AddressBookName = strName
    'Display message to user
    MsgBox ("The new address book name for the " & fldFolder.Name & strName & ".")
End Sub
AddressEntries Property

Returns the `AddressEntries` collection for the specified object.

`expression.AddressEntries`

`expression` Required. An expression that returns an `AddressList` object.
AddressEntry Property

Returns the AddressEntry object corresponding to the resolved recipient. Accessing the AddressEntry property forces resolution of an unresolved recipient name. If the name cannot be resolved, an error is returned. If the recipient is resolved, the Resolved property is True.

expression.AddressEntry

expression  Required. An expression that returns a Recipient object.
AddressLists Property

Returns an AddressLists collection representing a collection of the address lists available for this session. The AddressLists collection represents the root of the address book hierarchy for the current session. A particular AddressList object represents one of the available address books. The type of access you obtain depends on the access permissions granted to you by each individual address book provider.

expression.AddressLists

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

This Visual Basic for Applications example uses the `Add` method to add a new `AddressEntry` to the Personal Address Book. The `Name`, "John Q. Public" is necessary to avoid errors.

If this entry already exists, the code skips to `DialogBox:`. The `Details` method displays a dialog box that shows the information for this entry.

```vba
Set myOlApp = CreateObject("Outlook.Application")
Set myNamespace = myOlApp.GetNamespace("MAPI")
Set myAddrList = myNamespace.AddressLists("Personal Address Book")
Set myAddrEntries = myAddrList.AddressEntries
Set myEntry = myAddrEntries.Add("Microsoft Mail Address")
myEntry.Name = "John Q. Public"
On Error GoTo DialogBox
myEntry.Address = "someone@microsoft.com"
myEntry.Update
DialogBox:
    myEntry.Details
```
AllDayEvent Property

True if the appointment is an all-day event (as opposed to a specified time). Read/write Boolean.

expression.AllDayEvent

expression Required. An expression that returns an AppointmentItem object.
AlternateRecipientAllowed Property

True if the mail message can be forwarded. Read/write Boolean.

expression.AlternateRecipientAllowed

expression  Required. An expression that returns a MailItem object.
Anniversary Property

- Returns or sets a Date indicating the anniversary date for the contact. Read/write.

expression. Anniversary

expression Required. An expression that returns a ContactItem object.
**AnswerWizard Property**

Returns the `AnswerWizard` object for the application.

`expression.AnswerWizard()`

**expression**  Required. An expression that returns an `Application` object.
AppFolders Property

This property returns the "Application Folders" SyncObject. The SyncObject is where folders are automatically added when the InAppFolderSyncObject property of the MapiFolder object is set to True. The SyncObject allows users to synchronize Microsoft Outlook folders, address books, and folder home pages for offline use.

expression.AppFolders

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

The following example sets the SyncObject for the application folders and synchronizes the user's Inbox.

Public Sub SetAppfolders()

    Dim olApp As New Outlook.Application
    Dim nsp As Outlook.NameSpace
    Dim objSycs As Outlook.SyncObjects
    Dim objSyc As Outlook.SyncObject
    Dim mpfInbox As Outlook.MAPIFolder

    Set nsp = olApp.GetNamespace("MAPI")
    Set objSycs = nsp.SyncObjects
    Set objSyc = objSycs.AppFolders
    Set mpfInbox = nsp.GetDefaultFolder(olFolderInbox)

    mpfInbox.InAppFolderSyncObject = True

    objSyc.Start

End Sub
Application Property

- Returns an **Application** object that represents the parent application (Microsoft Outlook) for an object.

*expression*.**Application**

*expression*  Required. An expression that returns an Outlook object.
Example

This Visual Basic for Applications example uses the `Application` property to access Outlook and then creates a new `MailItem` and displays it.

```vba
Set myOlApp = CreateObject("Outlook.Application")
Set myItem = myOlApp.CreateItem(olMailItem)
myItem.Display
```

If you use VBScript, you do not use the `Application` property to retrieve the `Application` object. Instead, you reference the `Application` object directly. This example shows how to perform the same task using VBScript.

```vbs
Set myItem = Application.CreateItem(0)
myItem.Display
```
AppointmentItem Property

- Returns the AppointmentItem object that is the exception. Not valid for deleted appointments.

expression.AppointmentItem

expression  Required. An expression that returns an Exception object.
**Example**

This Visual Basic for Applications example uses **CreateItem** to create an **AppointmentItem** object. The **RecurrencePattern** is obtained for this item using the **GetRecurrencePattern** method. By setting the **RecurrencePattern** properties, **RecurrenceType**, **PatternStartDate**, and **PatternEndDate**, the appointments are now a recurring series that occur on a daily basis for the period of one year.

An **Exception** object is created when one instance of this recurring appointment is obtained using the **GetOccurrence** method and properties for this instance are altered. This exception to the series of appointments is obtained using the **GetRecurrencePattern** method to access the **Exceptions** collection associated with this series. Message boxes display the original **Subject** and **OriginalDate** for this exception to the series of appointments and the current date, time, and subject for this exception.

For a description of changes required for this example to work in VBScript, see the Note at the end of the example.

```vbnet
Public Sub cmdExample()
    Set myOlApp = New Outlook.Application
    Set myApptItem = myOlApp.CreateItem(olAppointmentItem)
    myApptItem.Start = #2/2/1998 3:00:00 PM#
    myApptItem.End = #2/2/1998 4:00:00 PM#
    myApptItem.Subject = "Meet with Boss"

    'Get the recurrence pattern for this appointment 'and set it so that this is a daily appointment 'that begins on 2/2/98 and ends on 2/2/99 'and save it.
    Set myRecurrPatt = myApptItem.GetRecurrencePattern
    myRecurrPatt.RecurrenceType = olRecursDaily
    myRecurrPatt.PatternStartDate = #2/2/1998#
    myRecurrPatt.PatternEndDate = #2/2/1999#
    myApptItem.Save

    'Access the items in the Calendar folder to locate 'the master AppointmentItem for the new series.
    Set myNameSpace = myOlApp.GetNamespace("MAPI")
    Set myFolder = myNameSpace.GetDefaultFolder(olFolderCalendar)
```
Set myItems = myFolder.Items
Set myApptItem = myItems("Meet with Boss")

'Get the recurrence pattern for this appointment
'and obtain the occurrence for 3/12/98.
myDate = #3/12/1998 3:00:00 PM#
Set myRecurrPatt = myApptItem.GetRecurrencePattern
Set myOddApptItem = myRecurrPatt.GetOccurrence(myDate)

'Save the existing subject. Change the subject and
'starting time for this particular appointment
'and save it.
saveSubject = myOddApptItem.Subject
myOddApptItem.Subject = "Meet NEW Boss"
newDate = #3/12/1998 3:30:00 PM#
myOddApptItem.Start = newDate
myOddApptItem.Save

'Get the recurrence pattern for the master
'AppointmentItem. Access the collection of
'exceptions to the regular appointments.
Set myRecurrPatt = myApptItem.GetRecurrencePattern
Set myException = myRecurrPatt.Exceptions.Item(1)

'Display the original date, time, and subject
'for this exception.
MsgBox myException.OriginalDate & "": " & saveSubject

'Display the current date, time, and subject
'for this exception.
MsgBox myException.AppointmentItem.Start & "": " & _
myException.AppointmentItem.Subject
End Sub

Note  For this example to work properly in VBScript, there are only a few
simple changes that need to be made in the code.

You don't have to retrieve the application as an object and you must use the
values of the constants, so:

Set myOlApp = New Outlook.Application
    Set myApptItem = myOlApp.CreateItem(olAppointmentItem)

becomes:

Set myApptItem = Application.CreateItem(1)
and

myRecurrPatt.RecurrenceType = olRecursDaily

becomes:

myRecurrPatt.RecurrenceType = 0

and

Set myFolder = myNamespace.GetDefaultFolder(olFolderCalendar)

becomes:

Set myFolder = myNamespace.GetDefaultFolder(9)
Assistant Property

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

Returns an Assistant object that represents the Microsoft Office Assistant.

expression.Assistant

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

- Returns or sets a **String** representing the name of the person who is the assistant for the contact. Read/write.

  \textit{expression.AssistantName}

  \textit{expression} Required. An expression that returns a **ContactItem** object.
AssistantTelephoneNumber Property

- Returns or sets a **String** representing the telephone number of the person who is the assistant for the contact. Read/write.

*expression*.AssistantTelephoneNumber

*expression*  Required. An expression that returns a ContactItem object.
Show All
Attachments Property

Returns an Attachments object that represents all the attachments for the specified item.

expression.Attachments

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

This Visual Basic for Applications example uses the **Remove** method to remove all attachments from a forwarded message before sending it on to John Y. Chen.

```vba
Set myOlApp = CreateObject("Outlook.Application")
Set myinspector = myOlApp.ActiveInspector
If Not TypeName(myinspector) = "Nothing" Then
    Set myitem = myinspector.CurrentItem.Forward
    Set myattachments = myitem.attachments
    If Not TypeName(myattachments) = "Nothing" Then
        While myattachments.Count > 0
            myattachments.Remove 1
        Wend
        myitem.Recipients.Add "John Y. Chen"
        myitem.Send
    End If
    MsgBox "The current item is not of the correct type"
Else
    MsgBox "There is no active Inspector"
End If
```

If you use VBScript, you do not create the **Application** object. This example shows how to perform the same task using VBScript.

```vbs
Set myinspector = Application.ActiveInspector
If Not TypeName(myinspector) = "Nothing" Then
    Set myitem = myinspector.CurrentItem.Forward
    Set myattachments = myitem.attachments
    If Not TypeName(myattachments) = "Nothing" Then
        While myattachments.Count > 0
            myattachments.Remove 1
        Wend
        myitem.Recipients.Add "John Y. Chen"
        myitem.Send
    End If
    MsgBox "The current item is not of the correct type"
Else
    MsgBox "There is no active Inspector"
End If
```
AutoForwarded Property

True if the mail message was automatically forwarded. Read/write Boolean.

expression.AutoForwarded

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

- 

Returns or sets a String representing the text of an automatic response for a Recipient. Read/write.

expression.AutoResponse

expression Required. An expression that returns a Recipient object.
BCC Property

Returns a String representing the display list of blind carbon copy (BCC) names for a MailItem. This property contains the display names only. The Recipients collection should be used to modify the BCC recipients. Read/write.

expression.BCC

expression  Required. An expression that returns a MailItem object.
BillingInformation Property

Returns or sets a String representing the billing information associated with the Outlook item. This is a free-form text field. Read/write.

expression.BillingInformation

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

Returns or sets a Date indicating the birthday for the contact. Read/write.

*expression*.Birthday

*expression*  Required. An expression that returns a ContactItem object.
Body Property

Returns or sets a **String** representing the clear-text body of the Outlook **item**. Read/write.

**Note** The **EditorType** property is not affected when you merely access the **Body** property of the item (as in MsgBox myItem.Body), but when you reset the **Body** property (as in myItem.Body = "This is a new body"), the **EditorType** reverts back to the user's default editor.

`expression.Body`

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

This VBScript example uses the **Open** event of an **item** to set its **Body** property.

Function Item_Open()
    Item.Body = "This is the message body."
End Function
BodyFormat Property

Returns or sets an OlBodyFormat constant indicating the format of the body text. The body text format determines the standard used to display the text of the message. Microsoft Outlook provides three body text format options: Plain Text, Rich Text and HTML. Read/write OlBodyFormat.

OlBodyFormat can be one of these OlBodyFormat constants.  
olFormatHTML  
olFormatPlain  
olFormatRichText  
olFormatUnspecified

expression.BodyFormat

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

All text formatting will be lost when the **BodyFormat** property is switched from RTF to HTML and vice-versa.

The property can not be set to **olFormatUnspecified**, however it will return **olFormatUnspecified** if the item has not yet been displayed.
Example

The following example creates a new MailItem object and sets the BodyFormat property to olFormatRichText. The Body text of the Mail item will now appear in Rich Text format.

Sub NewMail()
'Creates a new MailItem and modifies its properties

    Dim olApp As Outlook.Application
    Dim objMail As MailItem
    Set olApp = Outlook.Application
    'Create mail item
    Set objMail = olApp.CreateItem(olMailItem)

    With objMail
        .DownloadState = olHeaderOnly
        'Set body format to Rich Text
        .BodyFormat = olFormatRichText
        .Display
    End With

End Sub
Business2TelephoneNumber Property

- Returns or sets a String representing the second business telephone number for the contact. Read/write.

expression.Business2TelephoneNumber

expression Required. An expression that returns a ContactItem object.
BusinessAddress Property

Returns or sets a String representing the whole, unparsed business address for the contact. Read/write.

expression.BusinessAddress

desc Required. An expression that returns a ContactItem object.
BusinessAddressCity Property

Returns or sets a String representing the city name portion of the business address for the contact. Read/write.

expression.BusinessAddressCity

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

This property is parsed from the BusinessAddress property, but may be changed or entered independently should it be parsed incorrectly. Note that any such changes or entries to this property will be overwritten by any subsequent changes or entries to the BusinessAddress property.
BusinessAddressCountry Property

Returns or sets a String representing the country/region code portion of the business address for the contact. Read/write.

expression.BusinessAddressCountry

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

This property is parsed from the **BusinessAddress** property, but may be changed or entered independently should it be parsed incorrectly. Note that any such changes or entries to this property will be overwritten by any subsequent changes or entries to the **BusinessAddress** property.
**BusinessAddressPostalCode Property**

Returns or sets a `String` representing the postal code (zip code) portion of the business address for the contact. Read/write.

-expression.**BusinessAddressPostalCode**

`expression` Required. An expression that returns a [ContactItem](#) object.
Remarks

This property is parsed from the BusinessAddress property, but may be changed or entered independently should it be parsed incorrectly. Note that any such changes or entries to this property will be overwritten by any subsequent changes or entries to the BusinessAddress property.
**BusinessAddressPostOfficeBox Property**

Returns or sets a **String** representing the post office box number portion of the business address for the contact. Read/write.

*expression*.BusinessAddressPostOfficeBox

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

This property is parsed from the **BusinessAddress** property, but may be changed or entered independently should it be parsed incorrectly. Note that any such changes or entries to this property will be overwritten by any subsequent changes or entries to the **BusinessAddress** property.
**BusinessAddressState Property**

Returns or sets a `String` representing the state code portion of the business address for the contact. Read/write.

*expression*.BusinessAddressState

*expression*  Required. An expression that returns a `ContactItem` object.
Remarks

This property is parsed from the BusinessAddress property, but may be changed or entered independently should it be parsed incorrectly. Note that any such changes or entries to this property will be overwritten by any subsequent changes or entries to the BusinessAddress property.
**BusinessAddressStreet Property**

Returns or sets a `String` representing the street address portion of the business address for the contact. Read/write.

`expression.BusinessAddressStreet`

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

This property is parsed from the BusinessAddress property, but may be changed or entered independently should it be parsed incorrectly. Note that any such changes or entries to this property will be overwritten by any subsequent changes or entries to the BusinessAddress property.
**BusinessFaxNumber Property**

Returns or sets a **String** representing the business fax number for the contact. Read/write.

`expression.BusinessFaxNumber`

`expression` Required. An expression that returns a [ContactItem](#) object.
**BusinessHomePage Property**

Returns or sets a **String** representing the URL of the business Web page for the contact. Read/write.

*expression*.BusinessHomePage

*expression*  Required. An expression that returns a **ContactItem** object.
BusinessTelephoneNumber Property

Returns or sets a String representing the first business telephone number for the contact. Read/write.

expression.BusinessTelephoneNumber

expression  Required. An expression that returns a ContactItem object.
BusyStatus Property

Returns or sets an **OlBusyStatus** constant indicating the busy status of the user for the appointment. Read/write.

OlBusyStatus can be one of these OlBusyStatus constants.

- olBusy
- olFree
- olOutOfOffice
- olTentative

*expression*.BusyStatus

*expression* Required. An expression that returns an **AppointmentItem** object.
CallbackTelephoneNumber Property

Returns or sets a String representing the callback telephone number for the contact. Read/write.

expression.CallbackTelephoneNumber

expression  Required. An expression that returns a ContactItem object.
Caption Property

- Returns a **String** representing the window caption (title bar text) of an explorer or inspector window. Read-only.

`expression.Caption`

`expression` Required. An expression that returns an **Explorer** or **Inspector** object.
CardData Property

Returns or sets a `String` representing the text of the card data for the task. Read/write.

`expression.CardData`  

`expression`  Required. An expression that returns a `TaskItem` object.
CarTelephoneNumber Property

Returns or sets a String representing the car telephone number for the contact. Read/write.

expression.CarTelephoneNumber

expression Required. An expression that returns a ContactItem object.
Show All
Categories Property

Returns or sets a **String** representing the categories assigned to the Outlook item. Read/write.

**expression.Categories**

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

- **Returns** or sets a **String** representing the category assigned to the form description. Read/write.

  \[expression\].**Category**

  **expression**  Required. An expression that returns a **FormDescription** object.
CategorySub Property

Returns or sets a **String** representing the subcategory assigned to the form description. Read/write.

`expression.CategorySub`  

`expression` Required. An expression that returns a **FormDescription** object.
CC Property

Returns a String representing the display list of carbon copy (CC) names for a MailItem. This property contains the display names only. The Recipients collection should be used to modify the CC recipients. Read/write.

expression.CC

evaluation  Required. An expression that returns a MailItem object.
Children Property

Returns or sets a String representing the names of the children of the contact. Read/write.

expression.Children

expression Required. An expression that returns a ContactItem object.
Class Property

Returns an OlObjectClass constant indicating the object's class. Read-only.

OlObjectClass can be one of these OlObjectClass constants.

olAction
olActions
olAddressEntries
olAddressEntry
olAddressList
olAddressLists
olApplication
olAppointment
olAttachment
olAttachments
olAutoFormatRule
olAutoFormatRules
olCalendarViewField
olCalendarViewFields
olCardViewField
olCardViewFields
olClassCalendarView
olClassCardView
olClassIconView
olClassTableView
olClassTimelineView
olContact
olDistributionList
olDocument
olException
olExceptions
olExplorer
olExplorers
olFolder
olFolders
olFormDescription
olIconViewField
olIconViewFields
olInspector
olInspectors
olItemProperties
olItemProperty
olItems
olJournal
olLink
olLinks
olMail
olMeetingCancellation
olMeetingRequest
olMeetingResponseNegative
olMeetingResponsePositive
olMeetingResponseTentative
olNamespace
olNote
olObjects
olOutlookBarGroup
olOutlookBarGroups
olOutlookBarPane
olOutlookBarShortcut
olOutlookBarShortcuts
olOutlookBarStorage
olPages
olPanes
expression.Class

expression Required. An expression that returns an Outlook object.
Color Property

Returns or sets a Long indicating the color of the note. Can be one of the following OlNoteColor constants: olBlue, olGreen, olPink, olWhite, or olYellow. Read/write.

objNoteItem.Color

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

This example creates a new **NoteItem** and sets the color to blue.

```vbscript
Set myOlApp = CreateObject("Outlook.Application")
Set myItem = myOlApp.CreateItem(olNoteItem)
myItem.Color = olBlue
```

If you use VBScript, you do not create the **Application** object, and you cannot use named constants. This example shows how to perform the same task using VBScript.

```vbscript
Set myItem = Application.CreateItem(5)
myItem.Color = 0
```
COMAddIns Property

Returns a COMAddIns collection that represents all the Component Object Model (COM) add-ins currently loaded in Microsoft Outlook.

expression.COMAddIns

descriptor. COMAddIns

expression  Required. An expression that returns an Application object.
Example

This Microsoft Visual Basic example displays the number of COM add-ins currently connected.

Dim myOlApp As New Outlook.Application
Private Sub Command1_Click()
    MsgBox "There are " & _
        myOlApp.COMAddIns.Count & " COM add-ins."
End Sub

If you use VBScript, you do not declare an Application object variable. This example shows how to perform the same task using VBScript.

Sub Commandbutton1_Click()
    MsgBox "There are " & _
        Application.COMAddIns.Count & " COM add-ins."
End Sub
## CommandBars Property

- Returns a **CommandBars** collection object that represents all the menus and toolbars in the **Explorer** or **Inspector**.

```plaintext
expression.CommandBars
```

*expression* Required. An expression that returns an **Explorer** or **Inspector** object.
Comment Property

Returns or sets a `String` representing the text of a comment assigned to the form description. Read/write.

`expression.Comment`

`expression` Required. An expression that returns a `FormDescription` object.
Companies Property

Returns or sets a **String** representing the names of the companies associated with the Microsoft Outlook **item**. This is a free-form text field. Read/write.

\[ expression . Companies \]

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

Returns a **String** representing the concatenated company name and full name for the contact. Read-only.

*expression*.CompanyAndFullName

*expression* Required. An expression that returns a [ContactItem](#) object.
**CompanyLastFirstNoSpace Property**

Returns a **String** representing the company name for the contact followed by the concatenated last name, first name, and middle name with no space between the last and first names. This property is parsed from the **CompanyName**, **LastName**, **FirstName** and **MiddleName** properties. Read-only.

**Note** The **LastName**, **FirstName**, and **MiddleName** properties are themselves parsed from the **FullName** property.

**expression**.**CompanyLastFirstNoSpace**

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

**Note** The value of this property is only filled when its associated property (**FirstName**, **LastName**, **MiddleName**, **CompanyName**, and **Suffix**) contain Asian (DBCS) characters. If the corresponding field does not contain Asian characters, the property will be empty.
CompanyLastFirstSpaceOnly Property

Returns a String representing the company name for the contact followed by the concatenated last name, first name, and middle name with spaces between the last, first, and middle names. This property is parsed from the CompanyName, LastName, FirstName and MiddleName properties. Read-only.

**Note** The LastName, FirstName, and MiddleName properties are themselves parsed from the FullName property.

`expression.CompanyLastFirstSpaceOnly`

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

**Note** The value of this property is only filled when its associated property (FirstName, LastName, MiddleName, CompanyName, and Suffix) contain Asian (DBCS) characters. If the corresponding field does not contain Asian characters, the property will be empty.
CompanyMainTelephoneNumber Property

Returns or sets a String representing the company main telephone number for the contact. Read/write.

expression.CompanyMainTelephoneNumber

type expression Required. An expression that returns a ContactItem object.
Company Name Property

- Returns or sets a `String` representing the company name for the contact. Read/write.

`expression.CompanyName`

- `expression` Required. An expression that returns a `ContactItem` object.
Complete Property

True if the task is completed. Read/write Boolean.

expression.Complete

expression Required. An expression that returns a TaskItem object.
ComputerNetworkName Property

Returns or sets a String representing the name of the computer network for the contact. Read/write.

expression.ComputerNetworkName

expression Required. An expression that returns a ContactItem object.
ConferenceServerAllowExternal Property

Reserved for future use.
ConferenceServerPassword Property

Reserved for future use.
ContactName Property

Returns or sets a **String** representing the name of the person to contact for information regarding the custom form for this **FormDescription** object. Read/write.

`expression.ContactName`

*expression*  Required. An expression that returns a **FormDescription** object.
ContactNames Property

Returns a String representing the contact names associated with the journal entry. This property contains the display names for the contacts only. Use the Recipients object to modify the contents of this string. Read/write.

expression.ContactNames

expression Required. An expression that returns a JournalItem or TaskItem object.
Contents Property

Returns the OutlookBarStorage object for the specified Outlook Bar pane.

expression.Contents

expression  Required. An expression that returns an OutlookBarPane object.
Example

This Microsoft Visual Basic/Visual Basic for Applications example displays a message listing the groups in the Outlook Bar.

Dim myOlApp As New Outlook.Application
Dim myOlBar As Outlook.OutlookBarPane
Dim myOlGroups As Outlook.OutlookBarGroups
myMsg = "The groups in the Outlook Bar are:"
Set myOlBar = myOlApp.ActiveExplorer.Panes.Item("OutlookBar")
Set myOlGroups = myOlBar.Contents.Groups
For x = 1 To myOlGroups.Count
    myMsg = myMsg & Chr(13) & myOlGroups.Item(x)
Next x
MsgBox myMsg

If you use VBScript, you do not create the Application object. This example shows how to perform the same task using VBScript.

myMsg = "The groups in the Outlook Bar are:"
Set myOlBar = Application.ActiveExplorer.Panes.Item("OutlookBar")
Set myOlGroups = myOlBar.Contents.Groups
For x = 1 To myOlGroups.Count
    myMsg = myMsg & Chr(13) & myOlGroups.Item(x)
Next myOlGroups.Count
MsgBox myMsg
ConversationIndex Property

Returns a String representing the index of the conversation thread of the item. Read-only.

expression.ConversationIndex

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

Returns a String representing the topic of the conversation thread of the item. Read-only.

expression.ConversationTopic

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

Returns or sets an OlActionCopyLike constant indicating the property inheritance style to use for the action. The inheritance style is used when the action is executed to control how properties are copied to the new item created by the action. Read/write.

OlActionCopyLike can be one of these OlActionCopyLike constants.

- olForward
- olReply
- olReplyAll
- olReplyFolder
- olRespond

expression.Copy

expression Required. An expression that returns an Action object.
Count Property

Returns a **Long** indicating the count of objects in the specified collection.

*expression*.Count

*expression*  Required. An expression that returns a Microsoft Outlook collection object.
CreationTime Property

Returns a Date indicating the creation time for the Outlook item. This property corresponds to the MAPI property PR_CREATION_TIME. Read-only.

expression.CreationTime

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

Returns or sets a MAPIFolder object that represents the current folder displayed in the explorer. Use this property to change the folder the user is viewing.

expression.CurrentFolder

expression Required. An expression that returns an Explorer object.
Example

This Visual Basic for Applications example uses the `CurrentFolder` property to change the displayed folder to the user's Calendar folder.

```vba
Set myOlApp = CreateObject("Outlook.Application")
Set myNameSpace = myOlApp.GetNameSpace("MAPI")
Set myOlApp.ActiveExplorer.CurrentFolder = _
    myNameSpace.GetDefaultFolder(olFolderCalendar)
```

If you use VBScript, you do not create the `Application` object, and you cannot use named constants. This example shows how to perform the same task using VBScript.

```vbnet
Set myNameSpace = Application.GetNameSpace("MAPI")
Set Application.ActiveExplorer.CurrentFolder = _
    myNameSpace.GetDefaultFolder(9)
```
CurrentGroup Property

Returns or sets an OutlookBarGroup object that represents the group that is open in the specified Outlook Bar pane.

expression.CurrentGroup

expression  Required. An expression that returns an OutlookBarPane object.
Remarks

You must use the **Set** keyword to set this property, as shown in this example.

```vba
Set myOlPane.CurrentGroup = myOlGroup
```

Setting this property causes the **BeforeGroupSwitch** event to occur.
Show All
CurrentItem Property

Returns an **Object** representing the current **item** being displayed in the inspector.

**Note**  If there is no currently open item, an error will be returned.

`expression.CurrentItem`

`expression`  Required. An expression that returns an **Inspector** object.
Example

This Visual Basic for Applications example uses the `CurrentItem` property to obtain the current item that the user is viewing. If there is no currently open item, an error will be returned.

```vba
Set myOlApp = CreateObject("Outlook.Application")
On Error Resume Next
Set myItem = myOlApp.ActiveInspector.CurrentItem
```
CurrentUser Property

Returns the currently logged-on user as a Recipient object.

expression.CurrentUser

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

This Visual Basic for Applications example uses the `CurrentUser` property to obtain the name of the currently logged-on user and then displays a message box containing the name.

```vbnet
Set myOlApp = CreateObject("Outlook.Application")
Set myNameSpace = myOlApp.GetNamespace("MAPI")
MsgBox myNameSpace.CurrentUser.Name
```

If you use VBScript, you do not create the `Application` object. This example shows how to perform the same task using VBScript.

```vbs
Set myNameSpace = Application.GetNamespace("MAPI")
MsgBox myNameSpace.CurrentUser.Name
```
CurrentView Property

Returns or sets an Object (for the MAPIFolder object) or Variant (for the Explorer object) representing the current view. Read/write.

expression.CurrentView

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

When this property is set, two events occur: BeforeViewSwitch occurs before the actual view change takes place and can be used to cancel the change; ViewSwitch takes place after the change is effective.
Example

The following example sets the current view in the active explorer to Messages.

Dim myOlApp As New Outlook.Application
Dim myOlExp As Outlook.Explorer
Set myOlExp = myOlApp.ActiveExplorer
myOlExp.CurrentView = "Messages"

If you use VBScript, you do not create the Application object. This example shows how to perform the same task using VBScript.

Application.ActiveExplorer.CurrentView = "Messages"
CustomerID Property

- Returns or sets a String representing the customer ID for the contact. Read/write.

`objContactItem.CustomerID`

`objContactItem`  Required. An expression that returns a `ContactItem` object.
CustomViewsOnly Property

Returns or sets a Boolean that determines which views are displayed on the Views menu for a given folder. If set to the True, only user created-views will appear on the menu. Read/write.

expression.CustomViewsOnly

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

The following example prompts the user to select a view option. If the user chooses to view all views, the **CustomViewsOnly** property is set to **False**. If the user chooses to view only custom views, the **CustomViewsOnly** property is set to **True**. Once the property is changed, the outcome of the change is displayed for the user.

Sub SetCusView()
'Sets the CustomViewsOnly property depending on the user's response

    Dim olApp As Outlook.Application
    Dim nmsName As NameSpace
    Dim fldFolder As MAPIFolder
    Dim lngAns As Long

    Set olApp = Outlook.Application
    Set nmsName = olApp.GetNamespace("MAPI")
    Set fldFolder = nmsName.GetDefaultFolder(olFolderInbox)

    'Prompt user for input
    lngAns = MsgBox("Would you like to view only custom views in the view menu?", vbYesNo)

    Call SetVal(fldFolder, lngAns)

End Sub

Sub SetVal(ByRef fldFolder As MAPIFolder, ByVal lngAns As Long)
'Modifies the CustomViewsOnly property to display views on the Views menu

    If lngAns = vbYes Then
        fldFolder.CustomViewsOnly = True
    Else
        fldFolder.CustomViewsOnly = False
    End If

    'Display only custom views
    If lngAns = vbYes Then
    MsgBox "The " & fldFolder.Name & " folder will now display only custom views.
    Else
    MsgBox "The " & fldFolder.Name & " folder will now display all views.
    End If

End Sub
End Sub
DateCompleted Property

Returns or sets a Date indicating the completion date of the task. Read/write.

expression.DateCompleted

expression Required. An expression that returns a TaskItem object.
DayOfMonth Property

Returns or sets a Long indicating the day of the month on which the recurring appointment or task occurs. Read/write.

expression.DayOfMonth

type expression

Required. An expression that returns a RecurrencePattern object.
**DayOfWeekMask Property**

Returns or sets an **OldaysOfWeek** constant representing the mask for the days of the week on which the recurring appointment or task occurs. Monthly and yearly patterns are only valid for a single day. Weekly patterns are only valid as the **Or** of the **DayOfWeekMask**. Read/write.

*OldaysOfWeek* can be one of these *OldaysOfWeek* constants.

- olFriday
- olMonday
- olSaturday
- olSunday
- olThursday
- olTuesday
- olWednesday

*expression*.DayOfWeekMask

*expression* Required. An expression that returns a **RecurrencePattern** object.
Example

This Visual Basic for Applications example uses `GetRecurrencePattern` to obtain the `RecurrencePattern` object for the newly-created `AppointmentItem`. The properties, `RecurrenceType`, `DayOfWeekMask`, `PatternStartDate`, `Interval`, `PatternEndDate`, and `Subject` are set, the appointment is saved and then displayed with the pattern: "Occurs every 3 week(s) on Monday effective 1/21/98 until 12/21/2001 from 2:00 PM to 5:00 PM."

```vba
Set myOlApp = New Outlook.Application
Set myApptItem = myOlApp.CreateItem(olAppointmentItem)
Set myRecurrPatt = myApptItem.GetRecurrencePattern

myRecurrPatt.RecurrenceType = olRecursWeekly
myRecurrPatt.DayOfWeekMask = olMonday
myRecurrPatt.PatternStartDate = #1/21/98 2:00:00 PM#
myRecurrPatt.Interval = 3
myRecurrPatt.PatternEndDate = #12/21/2001 5:00:00 PM#
myApptItem.Subject = "Important Appointment"
myApptItem.Save
myApptItem.Display
```

If you use VBScript, you do not create the `Application` object, and you cannot use named constants. This example shows how to perform the same task using VBScript.

```vbs
Set myApptItem = Application.CreateItem(1)
Set myRecurrPatt = myApptItem.GetRecurrencePattern
myRecurrPatt.RecurrenceType = 1
myRecurrPatt.DayOfWeekMask = 2
myRecurrPatt.PatternStartDate = #1/21/98 2:00:00 PM#
myRecurrPatt.Interval = 3
myRecurrPatt.PatternEndDate = #12/21/2001 5:00:00 PM#
myApptItem.Subject = "Important Appointment"
myApptItem.Save
myApptItem.Display
```
DefaultItemType Property

Returns an **OlItemType** constant indicating the default Outlook item type contained in the folder. Read/write for the **Results** object; read-only for the **MAPIFolder** object.

OlItemType can be one of these OlItemType constants.

- **olAppointmentItem**
- **olContactItem**
- **olDistributionListItem**
- **olJournalItem**
- **olMailItem**
- **olNoteItem**
- **olPostItem**
- **olTaskItem**

*expression*.DefaultItemType

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

Returns a String representing the default message class for items in the folder. Read-only.

expression.DefaultMessageClass

expression  Required. An expression that returns a MAPIFolder object.
DeferredDeliveryTime Property

Returns or sets a Date indicating the date and time the mail message is to be delivered. This property corresponds to the MAPI property PR_DEFERRED_DELIVERY_TIME. Read/write.

expression.DeferredDeliveryTime

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

Returns an **OlTaskDelegationState** constant indicating the delegation state of the task. Read-only.

OlTaskDelegationState can be one of these OlTaskDelegationState constants.

- **olTaskDelegationAccepted**
- **olTaskDelegationDeclined**
- **olTaskDelegationUnknown**
- **olTaskNotDelegated**

**expression.DelegationState**

**expression**  Required. An expression that returns a **TaskItem** object.
Delegator Property

- Returns a String representing the display name of the delegator for the task. Read-only.

expression.Delegator

expression  Required. An expression that returns a TaskItem object.
DeleteAfterSubmit Property

True if a copy of the mail message is not saved upon being sent. False if a copy is saved. Read/write Boolean.

ttexpression.DeleteAfterSubmit

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

Indicates that the appointment became an exception because it was deleted from the recurring pattern. The Deleted property is True if the AppointmentItem was deleted. Read-only Boolean.

expression.Deleted

expression Required. An expression that returns an Exception object.
Department Property

Returns or sets a String representing the department name for the contact. Read/write.

expression.Department

expression Required. An expression that returns a ContactItem object.
Description Property

Returns or sets a **String** representing the description of the folder. This property corresponds to the MAPI property PR_COMMENT. Read/write.

`expression.Description`

`expression`  Required. An expression that returns a **MAPIFolder** object.
Dirty Property

Returns True if the contents of a custom property page have been altered. The ActiveX control that implements the PropertyPage object sets the value of this property, and Microsoft Outlook queries this in response to the OnStatusChange method of a PropertyPageSite object. Read-only Boolean.

expression.Dirty

expression Required. An expression that returns a PropertyPage object.
**Example**

This Visual Basic/Visual Basic for Applications example returns the value of the `Dirty` property as the value of a global variable.

```vbnet
Private Property Get PropertyPage_Dirty() As Boolean
    PropertyPage_Dirty = globDirty
End Property
```
**DisplayName Property**

For **Attachment** object:

Returns or sets a **String** representing the name, which need not be the actual file name, displayed below the icon representing the embedded attachment. This property corresponds to the MAPI property PR_DISPLAY_NAME. Read/write.

For **FormDescription** object:

Returns or sets a **String** representing the name of the form, which is what will be displayed in the **Choose Forms** dialog box. If both **FormDescription.Name** and **FormDescription.DisplayName** properties are empty, setting one will set the other. If one has been previously set, setting the other will not change the value. Read/write.

```expression.DisplayName```

**expression**  Required. An expression that returns an **Attachment** or **FormDescription** object.
**Example**

This Visual Basic for Applications example uses the *SaveAsFile* method to save the first attachment of the currently open item as a file in the My Documents folder, using the attachment's display name as the file name.

```vba
Set myOlApp = CreateObject("Outlook.Application")
Set myinspector = myOlApp.ActiveInspector
If Not TypeName(myinspector) = "Nothing" Then
  If TypeName(myinspector.CurrentItem) = "MailItem" Then
    Set myitem = myinspector.CurrentItem
    Set myAttachments = myitem.attachments
    myAttachments.Item(1).SaveAsFile "C:\" & _
    myAttachments.Item(1).DisplayName
  End If
MsgBox "The item is of the wrong type."
End If
```

If you use VBScript, you do not create the *Application* object. This example shows how to perform the same task using VBScript.

```vbnet
Set myinspector = Application.ActiveInspector
If Not TypeName(myinspector) = "Nothing" Then
  If TypeName(myinspector.CurrentItem) = "MailItem" Then
    Set myitem = myinspector.CurrentItem
    Set myAttachments = myitem.attachments
    myAttachments.Item(1).SaveAsFile "C:\" & _
    myAttachments.Item(1).DisplayName
  End If
MsgBox "The item is of the wrong type."
End If
```
DisplayType Property

Returns an OlDisplayType constant that describes the nature of the recipient. Read-only.

OlDisplayType can be one of these OlDisplayType constants.
  olAgent
  olDistList
  olForum
  olOrganization
  olPrivateDistList
  olRemoteUser
  olUser

You can use the DisplayType property to filter recipients. The DisplayType property corresponds to the MAPI property PR_DISPLAY_TYPE.

expression.DisplayType

expression  Required. An expression that returns an AddressEntry or Recipient object.
DLName Property

Returns or sets a **String** representing the display name of a distribution list. Read/write.

`expression.DLName`

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

This Microsoft Visual Basic/Visual Basic for Applications example creates a new distribution list and then prompts the user for the name.

Dim myOlApp As New Outlook.Application
Dim myDistList As Outlook.DistListItem
Set myDistList = myOlApp.CreateItem(olDistributionListItem)
myDistList.DLName = InputBox("Type the name of the new distribution list.")
myDistList.Display

If you use VBScript, you do not create the Application object, and you cannot use named constants. This example shows how to perform the same task using VBScript.

Set myDistList = Application.CreateItem(7)
myDistList.DLName = _
    InputBox("Type the name of the new distribution list.")
myDistList.Display
Show All
**DocPosted Property**

- **True** if the journalized *item* was posted as part of the journalized *session*. Read/write **Boolean**.

*expression*.**DocPosted**

*expression* Required. An expression that returns a **JournalItem** object.
Show All
DocPrinted Property

True if the journalized item was printed as part of the journalized session. Read/write Boolean.

expression.DocPrinted

expression Required. An expression that returns a JournalItem object.
**DocRouted Property**

- **True** if the journalized `item` was routed as part of the journalized `session`. Read/write **Boolean**.

  expression.<DocRouted>

  expression  Required. An expression that returns a `JournalItem` object.
Show All
DocSaved Property

True if the journalized item was saved as part of the journalized session. Read/write Boolean.

expression.DocSaved

expression Required. An expression that returns a JournalItem object.
DownloadState Property

- Returns or sets an OlDownloadState constant indicating the download state of the item. Read-only OlDownloadState.

OlDownloadState can be one of these OlDownloadState constants. olFullItem The entire item has been downloaded. olHeaderOnly Only the header has been downloaded.

expression.DownloadState

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

The following example searches through the user's Inbox for items that have not yet been downloaded. If any such items are found, a message is displayed to the user and the item is marked for download.

```vbnet
Public Sub DownloadStateTest()
'Tests items in the user's Inbox

    Dim outApp As Outlook.Application
    Dim mpfInbox As Outlook.MAPIFolder
    Dim obj As Object

    Set outApp = CreateObject("Outlook.Application")
    Set mpfInbox = outApp.GetNamespace("MAPI").GetDefaultFolder(olFo

    ' Loop all items in the Inbox Folder
    For i = 1 To mpfInbox.Items.Count
        Set obj = mpfInbox.Items.Item(i)
        'Verify if the state of the item is olHeaderOnly
        If obj.DownloadState = olHeaderOnly Then
            MsgBox ("This item has not been fully downloaded")
            'Mark the item to be downloaded
            obj.MarkForDownload = olMarkedForDownload
        End If
    Next

End Sub
```
DueDate Property

Returns or sets a Date indicating the due date for the task. Read/write.

expression.DueDate

expression  Required. An expression that returns a TaskItem object.
Duration Property

Returns or sets a `Long` indicating the duration (in minutes) of the appointment, journal entry, or recurrence pattern. For recurrences, this property is only valid for appointments. Read/write.

`expression.Duration`

`expression` Required. An expression that returns an `AppointmentItem`, `RecurrencePattern`, or `JournalItem` object.
Example

This Visual Basic for Applications example uses **CreateItem** to create an appointment and uses **MeetingStatus** to set the meeting status to "Meeting" to turn it into a meeting request with both a required and an optional attendee.

```vba
Set myOlApp = CreateObject("Outlook.Application")
Set myItem = myOlApp.CreateItem(olAppointmentItem)
myItem.MeetingStatus = olMeeting
myItem.Subject = "Strategy Meeting"
myItem.Location = "Conference Room B"
myItem.Start = #9/24/1997 1:30:00 PM#
myItem.Duration = 90
Set myRequiredAttendee = myItem.Recipients.Add("Nate Sun")
myRequiredAttendee.Type = olRequired
Set myOptionalAttendee = myItem.Recipients.Add("Kevin Kennedy")
myOptionalAttendee.Type = olOptional
Set myResourceAttendee = myItem.Recipients.Add("Conference Room B")
myResourceAttendee.Type = olResource
myItem.Display
```

If you use VBScript, you do not create the **Application** object, and you cannot use named constants. This example shows how to perform the same task using VBScript.

```vbs
Set myItem = Application.CreateItem(1)
myItem.MeetingStatus = 1
myItem.Subject = "Strategy Meeting"
myItem.Location = "Conference Room B"
myItem.Start = #9/24/97 1:30:00 PM#
myItem.Duration = 90
Set myRequiredAttendee = myItem.Recipients.Add("Nate Sun")
myRequiredAttendee.Type = 1
Set myOptionalAttendee = myItem.Recipients.Add("Kevin Kennedy")
myOptionalAttendee.Type = 2
Set myResourceAttendee = myItem.Recipients.Add("Conference Room B")
myResourceAttendee.Type = 3
myItem.Display
```
**EditorType Property**

Returns an **OlEditorType** constant indicating the type of editor. Read-only.

OlEditorType can be one of these OlEditorType constants.

- **olEditorHTML**
- **olEditorRTF**
- **olEditorText**
- **olEditorWord**

**Note** The **EditorType** property is not affected when you merely access the **Body** property of the item (as in MsgBox myItem.Body), but when you reset the **Body** property (as in myItem.Body = "This is a new body"), the **EditorType** reverts back to the user's default editor.

```
expression.EditorType
```

*expression* Required. An expression that returns an **Inspector** object.
Example

This VBScript example uses the **Open** event to access the **HTMLBody** property of an item. This sets the **EditorType** property of the item’s **Inspector** to **olEditorHTML**. When the item's **Body** property is set, the **EditorType** property is changed to the default. For example, if the default editor is set to RTF, the **EditorType** is set to **olEditorRTF**.

If this code is placed in the Script Editor of a form in design mode, the message boxes during run time will reflect the change in the **EditorType** as the body of the form changes. The final message box utilizes the **ScriptText** property to display all the VBScript code in the Script Editor.

```vb
Function Item_Open()
    'Set the HTMLBody of the item.
    Item.HTMLBody = "<HTML><H2>My HTML page.</H2><BODY>My body.</BODY></HTML>
    'Item displays HTML message.
    Item.Display
    'MsgBox shows EditorType is 2.
    MsgBox "HTMLBody EditorType is " & Item.GetInspector.EditorType
    'Access the Body and show
    'the text of the Body.
    MsgBox "This is the Body: " & Item.Body
    'After accessing, EditorType
    'is still 2.
    MsgBox "After accessing, the EditorType is " & Item.GetInspector.EditorType
    'Set the item's Body property.
    Item.Body = "Back to default body."
    'After setting, EditorType is
    'now back to the default.
    MsgBox "After setting, the EditorType is " & Item.GetInspector.EditorType
    'Access the items's
    'FormDescription object.
    Set myForm = Item.FormDescription
    'Display all the code
    'in the Script Editor.
    MsgBox myForm.ScriptText
End Function
```
Email1Address Property

- Returns or sets a String representing the e-mail address of the first e-mail entry for the contact. Read/write.

expression.Email1Address

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

This Visual Basic for Applications example sets "someone@microsoft.com" as the e-mail address for the first e-mail entry of a contact.

Set myOlApp = CreateObject("Outlook.Application")
Set myItem = myOlApp.CreateItem(olContactItem)
myItem.Email1Address = "someone@microsoft.com"

If you use VBScript, you do not create the Application object, and you cannot use named constants. This example shows how to perform the same task using VBScript.

Set myItem = Application.CreateItem(2)
myItem.Email1Address = "someone@microsoft.com"
Email1AddressType Property

Returns or sets a **String** representing the address type (such as EX or SMTP) of the first e-mail entry for the contact. This is a free-form text field, but it must match the actual type of an existing mail transport. Read/write.

_expression_.Email1AddressType

_expression_ Required. An expression that returns a `ContactItem` object.
Example

This Visual Basic for Applications example sets "SMTP" as the address type for the first e-mail entry of a contact.

```vba
Set myOlApp = CreateObject("Outlook.Application")
Set myItem = myOlApp.CreateItem(olContactItem)
myItem.Email1AddressType = "SMTP"
```

If you use VBScript, you do not create the `Application` object, and you cannot use named constants. This example shows how to perform the same task using VBScript.

```vba
Set myItem = Application.CreateItem(2)
myItem.Email1AddressType = "SMTP"
```
Email1DisplayName Property

Returns a String representing the display name of the first e-mail address for the contact. This property is set to the value of the FullName property by default. Read-only.

expression.Email1DisplayName

expression  Required. An expression that returns a ContactItem object.
Email1EntryID Property

Returns a String representing the entry ID of the first e-mail address for the contact. Read-only.

expression.Email1EntryID

description: Required. An expression that returns a ContactItem object.
Email2Address Property

Returns or sets a String representing the e-mail address of the second e-mail entry for the contact. Read/write.

expression.Email2Address

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

This Visual Basic for Applications example sets "someone@microsoft.com" as the e-mail address for the second e-mail entry of a contact.

```vba
Set myOlApp = CreateObject("Outlook.Application")
Set myItem = myOlApp.CreateItem(olContactItem)
myItem.Email2Address = "someone@microsoft.com"
```

If you use VBScript, you do not create the Application object, and you cannot use named constants. This example shows how to perform the same task using VBScript.

```vbs
Set myItem = Application.CreateItem(2)
myItem.Email2Address = "someone@microsoft.com"
```
Email2AddressType Property

Returns or sets a String representing the address type (such as EX or SMTP) of the second e-mail entry for the contact. This is a free-form text field, but it must match the actual type of an existing mail transport. Read/write.

expression.Email2AddressType

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

This Visual Basic for Applications example sets "SMTP" as the address type for the second e-mail entry of a contact.

```vba
Set myOlApp = CreateObject("Outlook.Application")
Set myItem = myOlApp.CreateItem(olContactItem)
myItem.Email2AddressType = "SMTP"
```

If you use VBScript, you do not create the `Application` object, and you cannot use named constants. This example shows how to perform the same task using VBScript.

```vbs
Set myItem = Application.CreateItem(2)
myItem.Email2AddressType = "SMTP"
```
Email2DisplayName Property

Returns a String representing the display name of the second e-mail entry for the contact. This property is set to the value of the FullName property by default. Read-only.

expression.Email2DisplayName

description

expression Required. An expression that returns a ContactItem object.
Email2EntryID Property

Returns a String representing the entry ID of the second e-mail entry for the contact. Read-only.

expression.Email2EntryID

expression Required. An expression that returns a ContactItem object.
Email3Address Property

Returns or sets a String representing the e-mail address of the third e-mail entry for the contact. Read/write.

expression.Email3Address

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

This Visual Basic for Applications example sets "someone@microsoft.com" as the e-mail address for the third e-mail entry of a contact.

Set myOlApp = CreateObject("Outlook.Application")
Set myItem = myOlApp.CreateItem(olContactItem)
myItem.\Field3Address = " someone@microsoft.com "

If you use VBScript, you do not create the Application object, and you cannot use named constants. This example shows how to perform the same task using VBScript.

Set myItem = Application.CreateItem(2)
myItem.\Field3Address = "someone@microsoft.com"
Email3AddressType Property

Returns or sets a String representing the address type (such as EX or SMTP) of the third e-mail entry for the contact. This is a free-form text field, but it must match the actual type of an existing mail transport. Read/write.

expression.Email3AddressType

expression Required. An expression that returns a ContactItem object.
**Example**

This Visual Basic for Applications example sets "SMTP" as the address type for the third e-mail entry of a contact.

```vba
Set myOlApp = CreateObject("Outlook.Application")
Set myItem = myOlApp.CreateItem(olContactItem)
myItem.Email3AddressType = "SMTP"
```

If you use VBScript, you do not create the `Application` object, and you cannot use named constants. This example shows how to perform the same task using VBScript.

```vbs
Set myItem = Application.CreateItem(2)
myItem.Email3AddressType = "SMTP"
```
Email3DisplayName Property

Returns a **String** representing the display name of the third e-mail entry for the contact. This property is set to the value of the **FullName** property by default. Read-only.

`expression.Email3DisplayName`  

`expression` Required. An expression that returns a **ContactItem** object.
Email3EntryID Property

Returns a **String** representing the entry ID of the third e-mail entry for the contact. Read-only.

*expression*.Email3EntryID

*expression* Required. An expression that returns a **ContactItem** object.
**Enabled Property**

*True* if the action is enabled in the application. Read/write *Boolean*.

*expression*.Enabled

*expression*  Required. An expression that returns an *Action* object.
End Property

- Returns or sets a Date indicating the end date and time of an appointment or journal entry. Read/write.

expression.End

expression Required. An expression that returns an AppointmentItem or a JournalItem object.
Example

This Visual Basic for Applications example uses **CreateItem** to create an **AppointmentItem** object. The **RecurrencePattern** is obtained for this item using the **GetRecurrencePattern** method. By setting the **RecurrencePattern** properties, **RecurrenceType**, **PatternStartDate**, and **PatternEndDate**, the appointments are now a recurring series that occur on a daily basis for the period of one year.

An **Exception** object is created when one instance of this recurring appointment is obtained using the **GetOccurrence** method and properties for this instance are altered. This exception to the series of appointments is obtained using the **GetRecurrencePattern** method to access the **Exceptions** collection associated with this series. Message boxes display the original **Subject** and **OriginalDate** for this exception to the series of appointments and the current date, time, and subject for this exception.

For a description of changes required for this example to work in VBScript, see the Note at the end of the example.

```vbnet
Public Sub cmdExample()
    Set myOlApp = New Outlook.Application
    Set myApptItem = myOlApp.CreateItem(olAppointmentItem)
    myApptItem.Start = #2/2/98 3:00:00 PM#
    myApptItem.End = #2/2/98 4:00:00 PM#
    myApptItem.Subject = "Meet with Boss"

    'Get the recurrence pattern for this appointment
    'and set it so that this is a daily appointment
    'that begins on 2/2/98 and ends on 2/2/99
    'and save it.
    Set myRecurrPatt = myApptItem.GetRecurrencePattern
    myRecurrPatt.RecurrenceType = olRecursDaily
    myRecurrPatt.PatternStartDate = #2/2/98#
    myRecurrPatt.PatternEndDate = #2/2/99#
    myApptItem.Save

    'Access the items in the Calendar folder to locate
    'the master AppointmentItem for the new series.
    Set myNamespace = myOlApp.GetNamespace("MAPI")
    Set myFolder = myNamespace.GetDefaultFolder(olFolderCalendar)
```
Set myItems = myFolder.Items
Set myApptItem = myItems("Meet with Boss")

' Get the recurrence pattern for this appointment
' and obtain the occurrence for 3/12/98.
myDate = #3/12/98 3:00:00 PM#
Set myRecurrPatt = myApptItem.GetRecurrencePattern
Set myOddApptItem = myRecurrPatt.GetOccurrence(myDate)

'Save the existing subject. Change the subject and
'starting time for this particular appointment
'and save it.
saveSubject = myOddApptItem.Subject
myOddApptItem.Subject = "Meet NEW Boss"
newDate = #3/12/98 3:30:00 PM#
myOddApptItem.Start = newDate
myOddApptItem.Save

' Get the recurrence pattern for the master
'AppointmentItem. Access the collection of
'exceptions to the regular appointments.
Set myRecurrPatt = myApptItem.GetRecurrencePattern
Set myException = myRecurrPatt.Exceptions.Item(1)

'Display the original date, time, and subject
'for this exception.
MsgBox myException.OriginalDate & "": " & saveSubject

'Display the current date, time, and subject
'for this exception.
MsgBox myException.AppointmentItem.Start & "": " & myException.AppointmentItem.Subject
End Sub

**Note**  For this example to work properly in VBScript, there are only a few simple changes that need to be made in the code.

You don't have to retrieve the application as an object and you must use the values of the constants, so:

Set myOlApp = New Outlook.Application
    Set myApptItem = myOlApp.CreateItem(olAppointmentItem)

becomes:

Set myApptItem = Application.CreateItem(1)
and

myRecurrPatt.RecurrenceType = olRecursDaily

becomes:

myRecurrPatt.RecurrenceType = 0

and

Set myFolder = myNamespace.GetDefaultFolder(olFolderCalendar)

becomes:

Set myFolder = myNamespace.GetDefaultFolder(9)
EndTime Property

Returns or sets a Date indicating the end time for a recurrence pattern. This property is only valid for appointments. Read/write.

expression.End

expression  Required. An expression that returns a RecurrencePattern object.
EntryID Property

Returns a **String** representing the unique entry ID of the object. This property corresponds to the MAPI property PR_ENTRYID. MAPI systems assign a permanent, unique ID string when an object is created that does not change from one MAPI **session** to another. The **EntryID** property is not set for an Outlook **item** until it is saved or sent. Also, the **EntryID** changes when an item is moved into another folder. Read-only.

`expression.EntryID`

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

This Visual Basic for Applications example uses the **EntryID** property to compare the entry ID of one message to the entry ID of a message returned by a search operation to determine whether the objects represent the same message.

```vba
Set myOlApp = CreateObject("Outlook.Application")
Set myNameSpace = myOlApp.GetNamespace("MAPI")
Set myContacts = myNameSpace.GetDefaultFolder(olFolderContacts)
Set myItem1 = myOlApp.CreateItem(olContactItem)
myItem1.FirstName = "Brent"
Set myItem2 = myContacts.Items.Find("[FileAs] = "Jones" and [FirstName] = "Brent"")
If Not TypeName(myItem2) = "Nothing" Then
    If myItem1.EntryID = myItem2.EntryID Then
        MsgBox "These two message objects refer to the same message."
    End If
Else
    MsgBox "The contact items were not found."
End If
```
Exceptions Property

- Returns the Exceptions collection for a specified series of recurring appointments.

expression.Exceptions

description  Required. An expression that returns a RecurrencePattern object.
Example

This Visual Basic for Applications example uses \texttt{CreateItem} to create an \texttt{AppointmentItem} object. The \texttt{RecurrencePattern} is obtained for this item using the \texttt{GetRecurrencePattern} method. By setting the \texttt{RecurrencePattern} properties, \texttt{RecurrenceType}, \texttt{PatternStartDate}, and \texttt{PatternEndDate}, the appointments are now a recurring series that occur on a daily basis for the period of one year.

An \texttt{Exception} object is created when one instance of this recurring appointment is obtained using the \texttt{GetOccurrence} method and properties for this instance are altered. This exception to the series of appointments is obtained using the \texttt{GetRecurrencePattern} method to access the \texttt{Exceptions} collection associated with this series. Message boxes display the original \texttt{Subject} and \texttt{OriginalDate} for this exception to the series of appointments and the current date, time, and subject for this exception.

For a description of changes required for this example to work in VBScript, see the Note at the end of the example.

\begin{verbatim}
Public Sub cmdExample()
    Set myOlApp = New Outlook.Application
    Set myApptItem = myOlApp.CreateItem(olAppointmentItem)
    myApptItem.Start = #2/2/98 3:00:00 PM#
    myApptItem.End = #2/2/98 4:00:00 PM#
    myApptItem.Subject = "Meet with Boss"

    'Get the recurrence pattern for this appointment
    'and set it so that this is a daily appointment
    'that begins on 2/2/98 and ends on 2/2/99
    'and save it.
    Set myRecurrPatt = myApptItem.GetRecurrencePattern
    myRecurrPatt.RecurrenceType = olRecursDaily
    myRecurrPatt.PatternStartDate = #2/2/98#
    myRecurrPatt.PatternEndDate = #2/2/99#
    myApptItem.Save

    'Access the items in the Calendar folder to locate
    'the master AppointmentItem for the new series.
    Set myNamespace = myOlApp.GetNamespace("MAPI")
    Set myFolder = myNamespace.GetDefaultFolder(olFolderCalendar)
\end{verbatim}
Set myItems = myFolder.Items
Set myApptItem = myItems("Meet with Boss")

'Get the recurrence pattern for this appointment
'and obtain the occurrence for 3/12/98.
myDate = #3/12/98 3:00:00 PM#
Set myRecurrPatt = myApptItem.GetRecurrencePattern
Set myOddApptItem = myRecurrPatt.GetOccurrence(myDate)

'Save the existing subject. Change the subject and
'starting time for this particular appointment
'and save it.
saveSubject = myOddApptItem.Subject
myOddApptItem.Subject = "Meet NEW Boss"
newDate = #3/12/98 3:30:00 PM#
myOddApptItem.Start = newDate
myOddApptItem.Save

'Get the recurrence pattern for the master
'AppointmentItem. Access the collection of
'exceptions to the regular appointments.
Set myRecurrPatt = myApptItem.GetRecurrencePattern
Set myException = myRecurrPatt.Exceptions.Item(1)

'Display the original date, time, and subject
'for this exception.
MsgBox myException.OriginalDate & ": " & saveSubject

'Display the current date, time, and subject
'for this exception.
MsgBox myException.AppointmentItem.Start & ": " & _
myException.AppointmentItem.Subject
End Sub

Note  For this example to work properly in VBScript, there are only a few
simple changes that need to be made in the code.

You don't have to retrieve the application as an object and you must use the
values of the constants, so:

Set myOlApp = New Outlook.Application
    Set myApptItem = myOlApp.CreateItem(olAppointmentItem)

becomes:

Set myApptItem = Application.CreateItem(1)
and

myRecurrPatt.RecurrenceType = olRecursDaily

becomes:

myRecurrPatt.RecurrenceType = 0

and

Set myFolder = myNamespace.GetDefaultFolder(olFolderCalendar)

becomes:

Set myFolder = myNamespace.GetDefaultFolder
ExpiryTime Property

Returns or sets a Date indicating the date and time at which the item becomes invalid and can be deleted. Read/write.

expression.ExpiryTime

expression Required. An expression that returns a MailItem, MeetingItem, or PostItem object.
Example

This VBScript example uses the Send event and sends an item with an automatic expiration date of one week.

```vbnet
Function Item_Send()
    Item.ExpiryTime = Date + 7
End Function
```
Explorers Property

Returns an **Explorers** collection object that contains the **Explorer** objects representing all open explorers.

**expression**.Explorers

**expression**  Required. An expression that returns an **Application** object.
Example

The following Microsoft Visual Basic example displays the number of explorer windows that are open.

```vbnet
Dim myOlApp As New Outlook.Application
Private Sub Command1_Click()
    MsgBox "There are " & _
End Sub

If you use VBScript, you do not declare an Application object variable. This example shows how to perform the same task using VBScript.

Sub CommandButton1_Click()
    MsgBox "There are " & _
End Sub
```
FileAs Property

- 

Returns or sets a **String** indicating the default keyword string assigned to the contact when it is filed. Read/write.

`expression.FileAs`

`expression`  Required. An expression that returns a **ContactItem** object.
FileName Property

Returns a String representing the file name of the attachment. Use this property in conjunction with the PathName property. Read-only.

expression.FileName

description

Required. An expression that returns an Attachment object.
Filter Property

The DASL statement used to restrict the search to a specified subset of data. This property is set by the Application object's AdvancedSearch method. Read-only String.

expression.Filter

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

The Filter property is set by the Filter argument when the Search object is first created.
Example

The following example creates a new Search object. The event subroutine fires after the search has finished and displays the Tag and Filter properties of the Search object as well as the results of the search.

Sub SearchInboxFolder()
'Searches the Inbox
    Dim objSch As Search
    Const strF As String = "urn:schemas:mailheader:subject = 'Office Christmas Party'"
    Const strS As String = "Inbox"
    Const strTag As String = "SubjectSearch"
    Set objSch = Application.AdvancedSearch(Scope:=strS, Filter:=strF, Tag:=s
End Sub

Use an AdvancedSearchComplete event subroutine to ensure the integrity of the data stored in the Search object.

Private Sub Application_AdvancedSearchComplete(ByVal SearchObject As Search
    Dim objRsts As Results
    MsgBox "The search " & SearchObject.Tag & "has completed. The fi
    SearchObject.Filter & "."
    Set objRsts = SearchObject.Results
    'Print out number in results collection
    Debug.Print objRsts.Count
    'Print out each member of results collection
    For Each Item In objRsts
        Debug.Print Item
    Next
End Sub
FirstName Property

Returns or sets a String representing the first name for the contact. Read/write.

expression.FirstName

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

This property is parsed from the **FullName** property, but may be changed or entered independently should it be parsed incorrectly. Note that any such changes or entries to this property will be overwritten by any subsequent changes of entries to **FullName**.
FlagDueBy Property

Returns or sets a **Date** specifying the date by which this mail message is due. This property is only valid if the **FlagStatus** property is also set for the message. This property corresponds to the MAPI property PR_REPLY_TIME. Read/write.

`expression.FlagDueBy`

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

Returns or sets a String indicating the requested action for the mail message. This is a free-form text field. This property is only valid if the FlagStatus property is also set for the message. Read/write.

expression.FlagRequest

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

Returns or sets an **OlFlagStatus** constant indicating the flag status for the mail message. Read/write.

OlFlagStatus can be one of these OlFlagStatus constants.

- `olFlagComplete`
- `olFlagMarked`
- `olNoFlag`

```plaintext
expression.FlagStatus
```

*expression*  Required. An expression that returns one of the items in the Applies To list.
FolderPath Property

Returns a read-only String that indicates the path of the current folder.

expression.FolderPath

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

The following example displays information about the current folder. The subroutine accepts a **MAPIFolder** object and displays the folder's name, path, and address book information.

```vba
Sub Folderpaths()
    Dim olApp As Outlook.Application
    Dim nmsName As NameSpace
    Dim fldFolder As MAPIFolder

    Set olApp = Outlook.Application
    'Create namespace reference
    Set nmsName = olApp.GetNamespace("MAPI")
    'create folder instance
    Set fldFolder = nmsName.GetDefaultFolder(olFolderInbox)
    'call sub program
    Call FolderInfo(fldFolder)

End Sub

Sub FolderInfo(ByVal fldFolder As MAPIFolder)
    'Displays information about a given folder
    MsgBox fldFolder.Name & "'s current path is " & fldFolder.FolderPath
        "The current address book name is " & fldFolder.AddressBookName

End Sub
```
Folders Property

Returns the `Folders` collection that represents all the folders contained in the specified folder or name space. The `NameSpace` object is the root of all the folders for the given name space.

`expression.Folders`

`expression` Required. An expression that returns a `MAPIFolder` object or a `NameSpace` object.
Example

This Visual Basic for Applications example uses the Add method to add the new folder named "My Contacts" to the current (default) Contacts folder.

```vba
Set myOlApp = CreateObject("Outlook.Application")
Set myNameSpace = myOlApp.GetNameSpace("MAPI")
Set myFolder = myNamespace.GetDefaultFolder(olFolderContacts)
Set myNewFolder = myFolder.Folders.Add("My Contacts")
```

If you use VBScript, you do not create the Application object, and you cannot use named constants. This example shows how to perform the same task using VBScript.

```vba
Set myNameSpace = Application.GetNameSpace("MAPI")
Set myFolder = myNamespace.GetDefaultFolder(10)
Set myNewFolder = myFolder.Folders.Add("My Contacts")
```

This Visual Basic for Applications example uses the Add method to add two new folders in the Tasks folder. The first folder, "Notes Folder", will contain note items. The second folder, "Contacts Folder", will contain contact items. If the folders already exist, a message box will inform the user.

```vba
Set myOlApp = CreateObject("Outlook.Application")
Set myNameSpace = myOlApp.GetNameSpace("MAPI")
Set myFolder = myNamespace.GetDefaultFolder(olFolderTasks)
On Error GoTo ErrorHandler
Set myNotesFolder = myFolder.Folders.Add("Notes Folder", olFolderNotes)
On Error GoTo ErrorHandler
Set myContactFolder = myFolder.Folders.Add("Contacts Folder", olFolderContacts)
Exit Sub
ErrorHandler:
    MsgBox "This folder already exists!"
```
FormDescription Property

- Returns the **FormDescription** object that represents the form description for the specified Microsoft Outlook **item**.

*expression*.**FormDescription**

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

Returns or sets a **String** representing the formula for the user property. Read/write.

```
expression.Formula
```

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

- Returns or sets a \textbf{String} representing the FTP site entry for the contact. Read/write.

\textit{expression}.FTPSite

\textit{expression} Required. An expression that returns a \textbf{ContactItem} object.
**FullName Property**

Returns or sets a String specifying the whole, unparsed full name for the contact. Read/write.

*expression.FullName*

*expression*  Required. An expression that returns a `ContactItem` object.
Remarks

This property is parsed into the **FirstName**, **LastName**, **MiddleName**, and **Suffix** properties, which may be changed or entered independently should they be parsed incorrectly. Note that any such changes or entries to the **FirstName**, **LastName**, **MiddleName** or **Suffix** properties will be overwritten by any subsequent changes or entries to **FullName**.
Example

This Visual Basic for Applications example uses the Restrict method to get all Inbox items dealing with Project X and moves them to the Project X folder.

Set myOlApp = CreateObject("Outlook.Application")
Set myNameSpace = myOlApp.GetNamespace("MAPI")
Set myFolder = myNameSpace.GetDefaultFolder(olFolderInbox)
Set myItems = myFolder.Items
Set myRestrictItems = myItems.Restrict _
_  ([Categories] = 'Project X')
For Each myItem In myRestrictItems
   myItem.Move myFolder.Folders("Project X")
Next

If you use VBScript, you do not create the Application object, and you cannot use named constants. This example shows how to perform the same task using VBScript.

Set myNameSpace = Application.GetNamespace("MAPI")
Set myFolder = myNameSpace.GetDefaultFolder(6)
Set myItems = myFolder.Items
Set myRestrictItems = myItems.Restrict _
_  ([Categories] = 'Project X')
For Each myItem In myRestrictItems
   myItem.Move myFolder.Folders("Project X")
Next

This Visual Basic for Applications example uses the Restrict method to apply a filter to the Contact items based on the item's LastModificationTime property.

Public Sub ContactDateCheck()
   Set myOlApp = CreateObject("Outlook.Application")
   Set myNameSpace = myOlApp.GetNamespace("MAPI")
   Set myContacts = myNameSpace.GetDefaultFolder(olFolderContacts).Items
   Set myItems = myContacts.Restrict([LastModificationTime] > '05/1
   For Each myItem In myItems
      MsgBox myItem.FullName & ": " & myItem.LastModificationTime
   Next
End Sub

The following Visual Basic for Applications example is the same as the example above, except that it demonstrates the use of a variable in the filter.
Public Sub ContactDateCheck()
    Set myOlApp = CreateObject("Outlook.Application")
    Set myNameSpace = myOlApp.GetNamespace("MAPI")
    Set myContacts = myNameSpace.GetDefaultFolder(olFolderContacts).Items
    DateStart = #6/11/97#
    DateToCheck$ = "[LastModificationTime] >= """ & DateStart & """
    Set myRestrictItems = myContacts.Restrict(DateToCheck$)
    For Each myItem In myRestrictItems
        MsgBox myItem.FullName & ": " & myItem.LastModificationTime
    Next
End Sub
**FullNameAndCompany Property**

Returns a **String** representing the full name and company of the contact by concatenating the values of the **FullName** and **CompanyName** properties. Read-only.

*expression*.FullNameAndCompany

*expression*  Required. An expression that returns a **ContactItem** object.
Gender Property

Returns or sets an OlGender constant indicating the gender of the contact. Read/write.

OlGender can be one of these OlGender constants.

  olFemale
  olMale
  olUnspecified

expression.Gender

expression  Required. An expression that returns a ContactItem object.
Show All
GetInspector Property

Returns an Inspector object that represents an inspector initialized to contain the specified item. This property is useful for returning a new Inspector object in which to display the item, as opposed to using the ActiveInspector method and setting the CurrentItem property.

expression.GetInspector

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

If the user’s default e-mail editor is Microsoft Word, and if the message format of the item being edited is plain text or HTML, GetInspector produces an error and returns Nothing.
Example

This Visual Basic for Applications example uses the `GetInspector` property to return a new, inactive inspector for `myItem`, and then displays it.

```vba
Set myOlApp = CreateObject("Outlook.Application")
Set myItem = myOlApp.CreateItem(olMailItem)
Set myInspector = myItem.GetInspector
myInspector.Display
```

If you use VBScript, you do not create the `Application` object, and you cannot use named constants. This example shows how to perform the same task using VBScript.

```vbs
Set myItem = Application.CreateItem(0)
Set myInspector = myItem.GetInspector
myInspector.Display
```
GovernmentIDNumber Property

- Returns or sets a String representing the government ID number for the contact. Read/write.

$expression.GovernmentIDNumber$

$expression$ Required. An expression that returns a ContactItem object.
Groups Property

- Returns an OutlookBarGroups object representing the set of groups in an Outlook Bar.

expression.Groups

expression  Required. An expression that returns an OutlookBarStorage object.
Example

The following Microsoft Visual Basic/Visual Basic for Applications example displays the number of groups in the Outlook Bar.

```
Dim myOlApp As New Outlook.Application
Dim myOlBar As Outlook.OutlookBarPane
Set myOlBar = myOlApp.ActiveExplorer.Panes.Item("OutlookBar")
myCount = myOlBar.Contents.Groups.Count
MsgBox "There are " & myCount & " groups in the Outlook Bar"
```

If you use VBScript, you do not create the Application object. This example shows how to perform the same task using VBScript.

```
Set myOlBar = Application.ActiveExplorer.Panes.Item("OutlookBar")
myCount = myOlBar.Contents.Groups.Count
MsgBox "There are " & myCount & " groups in the Outlook Bar"
```
HasAttachment Property

True (default) if the remote item has an attachment associated with it. Read-only Boolean.

expression.HasAttachment

expression Required. An expression that returns a RemoteItem object.
Height Property

Returns or sets a Long specifying the height (in pixels) of the explorer, inspector, or note window. Read/write.

expression.Height

expression Required. An expression that returns an Explorer, Inspector, or NoteItem object.
Hidden Property

- True to cause the specified custom form to be hidden. It will not appear on the menu or in the Choose Form dialog box and will be used only if it is designated as the response form from another custom form. The default value is False. Read/write Boolean.

expression.Hidden

expression  Required. An expression that returns a FormDescription object.
Hobby Property

- Returns or sets a **String** representing the hobby for the contact. Read/write.

  _expression_.Hobby

  expression  Required. An expression that returns a **ContactItem** object.
Home2TelephoneNumber Property

Returns or sets a String representing the second home telephone number for the contact. Read/write.

expression.Home2TelephoneNumber

expression Required. An expression that returns a ContactItem object.
HomeAddress Property

Returns or sets a String representing the full, unparsed text of the home address for the contact. Read/write.

Expression.HomeAddress

Expression Required. An expression that returns a ContactItem object.
HomeAddressCity Property

- Returns or sets a String representing the city portion of the home address for the contact. Read/write.

expression.HomeAddressCity

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

This property is parsed from the HomeAddress property, but may be changed or entered independently should it be parsed incorrectly. Note that any such changes or entries to this property will be overwritten by any subsequent changes or entries to HomeAddress.
HomeAddressCountry Property

- Returns or sets a String representing the country/region portion of the home address for the contact. Read/write.

expression.HomeAddressCountry

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

This property is parsed from the **HomeAddress** property, but may be changed or entered independently should it be parsed incorrectly. Note that any such changes or entries to this property will be overwritten by any subsequent changes or entries to **HomeAddress**.
HomeAddressPostalCode Property

Returns or sets a **String** representing the postal code portion of the home address for the contact. Read/write.

`expression.HomeAddressPostalCode`

`expression` Required. An expression that returns a **ContactItem** object.
Remarks

This property is parsed from the HomeAddress property, but may be changed or entered independently should it be parsed incorrectly. Note that any such changes or entries to this property will be overwritten by any subsequent changes or entries to HomeAddress.
HomeAddressPostOfficeBox Property

Returns or sets a **String** the post office box number portion of the home address for the contact. Read/write.

`expression.HomeAddressPostOfficeBox`

`expression`  Required. An expression that returns a **ContactItem** object.
Remarks

This property is parsed from the **HomeAddress** property, but may be changed or entered independently should it be parsed incorrectly. Note that any such changes or entries to this property will be overwritten by any subsequent changes or entries to **HomeAddress**.
HomeAddressState Property

Returns or sets a `String` representing the state portion of the home address for the contact. Read/write.

`expression.HomeAddressState`

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

This property is parsed from the HomeAddress property, but may be changed or entered independently should it be parsed incorrectly. Note that any such changes or entries to this property will be overwritten by any subsequent changes or entries to HomeAddress.
HomeAddressStreet Property

Returns or sets a String representing the street portion of the home address for the contact. Read/write.

expression.HomeAddressStreet

description Required. An expression that returns a ContactItem object.
Remarks

This property is parsed from the HomeAddress property, but may be changed or entered independently should it be parsed incorrectly. Note that any such changes or entries to this property will be overwritten by any subsequent changes or entries to HomeAddress.
HomeFaxNumber Property

Returns or sets a **String** representing the home fax number for the contact. Read/write.

```
expression.HomeFaxNumber
```

*expression* Required. An expression that returns a **ContactItem** object.
HomeTelephoneNumber Property

Returns or sets a **String** representing the first home telephone number for the contact. Read/write.

`expression.HomeTelephoneNumber`

`expression` Required. An expression that returns a **ContactItem** object.
Show All
HTMLBody Property

Returns or sets a **String** representing the HTML body of the specified item. The **HTMLBody** property should be an HTML syntax string. Read/write.

Setting the **HTMLBody** property sets the **EditorType** property of the item's **Inspector** to **olEditorHTML**.

Setting the **HTMLBody** property will always update the **Body** property immediately.

Setting the **Body** property will clear the contents of the **HTMLBody** property on HTML aware stores.

**Note** The **EditorType** property is not affected when you merely access the **Body** property of the item (as in `MsgBox myItem.Body`), but when you reset the **Body** property (as in `myItem.Body = "This is a new body"`), the **EditorType** reverts back to the user's default editor.

**expression**.**HTMLBody**

**expression**  Required. An expression that returns a **PostItem** or **MailItem** object.
Example

This VBScript example uses the **Open** event to access the **HTMLBody** property of an item. This sets the **EditorType** property of the item’s **Inspector** to **olEditorHTML**. When the item's **Body** property is set, the **EditorType** property is changed to the default. For example, if the default editor is set to RTF, the **EditorType** is set to **olEditorRTF**.

If this code is placed in the Script Editor of a form in design time, the message boxes during run time will reflect the change in the **EditorType** as the body of the form changes. The final message box utilizes the **ScriptText** property to display all the VBScript code in the Script Editor.

```vbnet
Function Item_Open()
    ' Set the HTMLBody of the item.
    Item.HTMLBody = "<HTML><H2>My HTML page.</H2><BODY>My body.</BODY></HTML>
    ' Item displays HTML message.
    Item.Display
    ' MsgBox shows EditorType is 2.
    MsgBox "HTMLBody EditorType is " & Item.GetInspector.EditorType
    ' Access the Body and show
    ' the text of the Body.
    MsgBox "This is the Body: " & Item.Body
    ' After accessing, EditorType
    ' is still 2.
    MsgBox "After accessing, the EditorType is " & Item.GetInspector.EditorType
    ' Set the item's Body property.
    Item.Body = "Back to default body."
    ' After setting, EditorType is
    ' now back to the default.
    MsgBox "After setting, the EditorType is " & Item.GetInspector.EditorType
    ' Access the items's
    ' FormDescription object.
    Set myForm = Item.FormDescription
    ' Display all the code
    ' in the Script Editor.
    MsgBox myForm.ScriptText
End Function
```
 HTMLDocument Property

Returns an HTMLDocument object that specifies the HTML object Model associated with the HTML document in the current view (assuming one exists).

expression. HTMLDocument

expression Required. An expression that returns an Explorer object.
Remarks

In order to use this property you must set the homepage property of your **MAPIfolder** to a Web page, or set the **WebViewURL** property of the **MAPIFolder** object to a web page.
Example

The following example accesses the Outlook View Control.

Sub GetHTML()
'Returns the Outlook View control

    Dim objExp As Explorer
    Set objExp = Application.ActiveExplorer
    Dim HTMLDoc As HTMLDocument
    Set HTMLDoc = objExp.HTMLDocument
    Dim objVC As ViewControl
    Set objVC = HTMLDoc.All.Tags("object").Item(0).Object

End Sub
HTMLEditor Property

Returns an Object representing the HTML Document Object Model of the message being displayed. The HTML Document Object Model is defined by Microsoft Internet Explorer and is the same one used for Dynamic HTML. This object may be temporary and should not be stored for later use. Read-only.

The HTMLEditor property is only valid if the EditorType property of the item's associated Inspector is set to olEditorHTML.

expression.HTMLEditor

expression  Required. An expression that returns an Inspector object.
Example

The following VBScript example uses the Click event of a CommandButton control named "CommandButton1" to demonstrate the listing of all HTML elements.

Sub CommandButton1_Click()
    Dim i 'As Integer
    Dim strHTMLType 'As String
    Dim strHTMLText 'As String
    Dim NL 'As String

    NL = chr(10) & chr(13)

    Set myInspector = Item.GetInspector
    Set myIExplorer = myInspector.HTMLElement
    If myIExplorer.ReadyState <> "complete" Then
        'Test for complete loading of HTML doc
        For i = 0 To myIExplorer.All.Length - 1
            strHTMLType = TypeName(myIExplorer.All.Item(i))
            On Error Resume Next
        Next
        'because not all elements support OuterHTML
        strHTMLText = ": " & NL & myIExplorer.All.Item(i).outerHTML
        On Error GoTo 0
        MsgBox strHTMLType & strHTMLText
        strHTMLText = ""
    Next
End If
End Sub
Icon Property

Returns or sets a String representing the file name of the icon to be displayed for the form. Read/write.

expression.Icon

expression Required. An expression that returns a FormDescription object.
ID Property

Returns a String representing the unique identifier for the object. The transport provider assigns a permanent, unique string ID property when an individual member object is created. These identifiers do not change from one session to another. Read-only.

expression.ID

expression Required. An expression that returns an AddressEntry or AddressList object.
IMAddress Property

- Returns or sets a String that represents a contact's Microsoft Instant Messenger address. Read/write.

expression.IMAddress

descriptor expression  Required. An expression that returns a ContactItem object.
Remarks

Unlike the **Recipients** or **To** properties, there is no way to verify that the **IMAddress** property contains a valid address.
**Example**

The following example creates a new contact and prompts the user to enter an Instant Messenger address for the contact.

```vbs
Sub SetImAddress()
'Sets a new IM Address
    Dim olApp As Outlook.Application
    Dim objNewContact As ContactItem

    Set olApp = Outlook.Application
    Set objNewContact = olApp.CreateItem(olContactItem)
    objNewContact.IMAddress = _
        InputBox("Enter the new contact's Microsoft Instant Messenger address")
    objNewContact.Save
End Sub
```
Importance Property

Returns or sets an OlImportance constant indicating the relative importance level for the Outlook item. This property corresponds to the MAPI property PR_IMPORTANCE. Read/write.

OlImportance can be one of these OlImportance constants.

- olImportanceHigh
- olImportanceLow
- olImportanceNormal

expression.Importance

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

Returns or sets a **Boolean** that determines if the specified folder will be synchronized with the mail server. If **True**, this folder will be synchronized when the "application folders" **SyncObject** is synchronized. If **False**, the folder will not synchronize. Read/write.

*expression*.InAppFolderSyncObject

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

This is equivalent to clicking the check box for this folder in the "application folders" group of the Send/Receive dialog box.

If this property is set to True, and the "application folders" SyncObject does not already exist, a SyncObject will be automatically created.
Example

The following example sets the current **MAPIFolder** to synchronize when the "Application Folders" **SyncObject** object is synchronized. Here, the **InAppFolderSyncObject** property is used in conjunction with the **AppFolders** property of the **SyncObject**.

```vba
Public Sub appfolders()
    Dim olApp As New Outlook.Application
    Dim nsp As Outlook.NameSpace
    Dim sycs As Outlook.SyncObjects
    Dim syc As Outlook.SyncObject
    Dim mpfInbox As Outlook.MAPIFolder

    Set nsp = olApp.GetNamespace("MAPI")
    Set sycs = nsp.SyncObjects
    'Return the application folder SyncObject
    Set syc = sycs.appfolders
    'Get the Inbox folder
    Set mpfInbox = nsp.GetDefaultFolder(olFolderInbox)
    'Set the Inbox Folder to be synchronized when the application
    'folder's SyncObject is synchronized
    mpfInbox.InAppFolderSyncObject = True
    'Star the synchronization
    syc.Start
End Sub
```
IncludeRecurrences Property

True if the Items collection should include recurrence patterns. This property only has an effect if the Items collection contains appointments and is not sorted by any property other than Start in ascending order. The default value is False. Read/write Boolean.

expression.IncludeRecurrences

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

Use this property when you want to retrieve all appointments for a given date, where recurring appointments would not normally appear because they are not associated with any specific date. If the collection includes recurring appointments with no end date, setting the property to True may cause the collection to be of infinite count. Be sure to include a test for this in any loop.
Index Property

Returns a **Long** indicating the position of the object within the collection. The **Index** property is only valid during the current **session** and can change as objects are added to and deleted from the collection. The first object in the collection has an **Index** value of 1. Read-only.

`expression.Index`

*expression*  Required. An expression that returns an **AddressList**, **Attachment**, or **Recipient** object.
Initials Property

- Returns or sets a String representing the initials for the contact. Read/write.

expression.Initials

expression Required. An expression that returns a ContactItem object.
Inspectors Property

Returns an **Inspectors** collection object that contains the **Inspector** objects representing all open inspectors.

`expression.Inspectors`

*expression*  Required. An expression that returns an **Application** object.
Example

This Microsoft Visual Basic example uses the **Inspectors** property and the **Count** property and **Item** method of the **Inspectors** object to display the captions of all inspector windows.

```
Dim myOlApp As New Outlook.Application
Private Sub Command1_Click()
    If myOlApp.Inspectors.Count > 0 Then
        For x = 1 To myOlApp.Inspectors.Count
            MsgBox Inspectors.Item(x).Caption
        Next x
    Else
        MsgBox "There are no inspector windows open."
    End If
End Sub
```

If you use VBScript, you do not declare an **Application** object variable. This example shows how to perform the same task using VBScript.

```
Private Sub Command1_Click()
    If Application.Inspectors.Count > 0 Then
        For x = 1 To myOlApp.Inspectors.Count
            MsgBox Inspectors.Item(x).Caption
        Next x
    Else
        MsgBox "There are no inspector windows open."
    End If
End Sub
```
Instance Property

Returns or sets a `Long` specifying the count for which the recurrence pattern is valid for a given interval. This property is only valid for recurrences of the `olRecursMonthNth` and `olRecursYearNth` type and allows the definition of a recurrence pattern that is only valid for the Nth occurrence, such as "the 2nd Sunday in March" pattern. The count is set numerically: 1 for the first, 2 for the second, and so on through 5 for the last. Values greater than 5 will generate errors when the pattern is saved. Read/write.

`expression.Instance`

`expression` Required. An expression that returns a `RecurrencePattern` object.
InternetCodepage Property

Returns or sets a Long that determines the Internet code page used by the item. The Internet code page defines the text encoding scheme used by the item. Read/write.

\[ \text{expression}.\text{InternetCodepage} \]

\text{expression} Required. An expression that returns one of the objects in the Applies To list.
Example

The following example searches through the user's Inbox and displays the sender names for all mail items with the Internet code page value 1256. This value corresponds to the Internet code page value for Arabic text. If no mail items are found, a message is displayed to the user.

Sub FindArabicUsers()
'Displays information about an Internet code page

    Dim olApp As Outlook.Application
    Dim objMail As MailItem
    Dim objItems As Items
    Dim strUsers As String

    Const cstArabic As String = "1256"

    Set olApp = Outlook.Application
    Set objItems = _
        olApp.GetNamespace("MAPI").GetDefaultFolder(olFolderInbox).Items
    For Each objMail In objItems
        If objMail.InternetCodepage = cstArabic Then
            strUser = strUser & objMail.SenderName & vbCrLf
        End If
    Next objMail

    If strUsers = "" Then
        MsgBox "There are no Mail items in the Inbox from users with " & _
            "Internet code page."
    Else
        MsgBox "The following users use the specified foreign Intern " & _
            "et code page:"
            & vbCrLf & vbCrLf & strUser
    End If

End Sub
InternetFreeBusyAddress Property

Returns or sets a String corresponding to the Address box on the Details tab for a contact. This box can contain the URL location of the user's free-busy information in vCard Free-Busy standard format. Read/write.

expression.InternetFreeBusyAddress

description Required. An expression that returns a ContactItem object.
**Interval Property**

Returns or sets a **Long** specifying the number of units of a given recurrence type between occurrences. For example, setting the **Interval** property to 2 and the **RecurrenceType** property to "Weekly" would cause the pattern to occur every second week. Read/write.

**Note** The **Interval** property must be set before setting **PatternEndDate**. Also, the **Interval** property is not valid for yearly recurrence patterns.

**expression.Interval**

**expression**  Required. An expression that returns a **RecurrencePattern** object.
Example

This Visual Basic for Applications example uses GetRecurrencePattern to obtain the RecurrencePattern object for the newly-created AppointmentItem. The properties, RecurrenceType, DayOfWeekMask, PatternStartDate, Interval, PatternEndDate, and Subject are set, the appointment is saved and then displayed with the pattern: "Occurs every 3 week(s) on Monday effective 1/21/98 until 12/21/2001 from 2:00 PM to 5:00 PM."

Set myOlApp = New Outlook.Application
Set myApptItem = myOlApp.CreateItem(olAppointmentItem)
Set myRecurrPatt = myApptItem.GetRecurrencePattern

myRecurrPatt.RecurrenceType = olRecursWeekly
myRecurrPatt.DayOfWeekMask = olMonday
myRecurrPatt.PatternStartDate = #1/21/98 2:00:00 PM#
myRecurrPatt.Interval = 3
myRecurrPatt.PatternEndDate = #12/21/2001 5:00:00 PM#
myApptItem.Subject = "Important Appointment"
myApptItem.Save
myApptItem.Display

If you use VBScript, you do not create the Application object, and you cannot use named constants. This example shows how to perform the same task using VBScript.

Set myApptItem = Application.CreateItem(1)
Set myRecurrPatt = myApptItem.GetRecurrencePattern
myRecurrPatt.RecurrenceType = 1
myRecurrPatt.DayOfWeekMask = 2
myRecurrPatt.PatternStartDate = #1/21/98 2:00:00 PM#
myRecurrPatt.Interval = 3
myRecurrPatt.PatternEndDate = #12/21/2001 5:00:00 PM#
myApptItem.Subject = "Important Appointment"
myApptItem.Save
myApptItem.Display
IsConflict Property

Returns a Boolean that determines if the item is in conflict. Whether or not an item is in conflict is determined by the state of the application. For example, when a user is offline and tries to access an online folder the action will fail. In this scenario, the IsConflict property will return True. Read-only.

expression.IsConflict

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

If True, the specified item is in conflict.
Example

The following example creates a new message and attempts to send it. If the IsConflict property returns True, the item will not be sent.

Sub NewMail()
'Creates and attempts to send a new e-mail message

    Dim olApp As Outlook.Application
    Dim objNewMail As MailItem

    Set olApp = Outlook.Application
    Set objNewMail = olApp.CreateItem(olMailItem)
    objNewMail.Body = "This e-mail message was created automatically"
    objNewMail.To = "Jeff Smith"

    If objNewMail.IsConflict = False Then
        objNewMail.Send
    Else
        MsgBox "Conflict: Cannot send e-mail"
    End If

End Sub
ISDNNumber Property

Returns or sets a String representing the ISDN number for the contact. Read/write.

expression.ISDNNumber

description Required. An expression that returns a ContactItem object.
IsOnlineMeeting Property

*True* if this is an online meeting. Read/write *Boolean*.

`expression.IsOnlineMeeting`

`expression` Required. An expression that returns an *AppointmentItem* object.
IsReadOnly Property

Indicates that the AddressList object cannot be modified. The IsReadOnly property refers to adding and deleting the entries in the address book container represented by the AddressList object. The property is True if no entries can be added or deleted. The property is False if the container can be modified, that is, if address entries can be added to and deleted from the container.

The IsReadOnly property refers to the address book entries in the context of the address book container. It does not indicate whether the contents of the individual entries themselves can be modified.

Read-only Boolean.

evaluation.IsReadOnly
evaluation  Required. An expression that returns an AddressList object.
**IsRecurring Property**

*True* if the appointment or task is a recurring appointment or task. When the `GetRecurrencePattern` method is used with an `AppointmentItem` or `TaskItem` object, this property is set to *True*. Read-only *Boolean*.

`expression.IsRecurring`

`expression` Required. An expression that returns a `TaskItem` or an `AppointmentItem` object.
IsSynchronous Property

Returns a Boolean indicating whether the search is synchronous.

expression.IsSynchronous

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

If the search is synchronous, the user's computer will wait until the search has completed. Conversely, if the search is asynchronous, the search could still execute when the code has finished running. In this case, use the Search object's Stop method to halt the search.

In order to get meaningful results from an asynchronous search, use the AdvancedSearchComplete event to notify you when the search has finished.
IsUserProperty Property

Returns a **Boolean** value that indicates if the item property is a custom property created by the user. Read-only.

`expression.IsUserProperty`

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

This collection is 0 based. In other words, the first object in the collection is accessed with an index value of (0) zero.
Example

The following example displays the names of all properties created by the user. The subroutine DisplayUserProps accepts an ItemProperties collection and searches through it, displaying the names of all ItemProperty objects where the IsUserProperty value is True.

Sub ItemProperty()
'Creates a new mail item and access it's properties

    Dim olApp As Outlook.Application
    Dim objMail As MailItem
    Dim objitems As ItemProperties

    Set olApp = Outlook.Application
    'Create the mail item
    Set objMail = olApp.CreateItem(olMailItem)
    'Create a reference to the item properties collection
    Set objitems = objMail.ItemProperties
    'Create a reference to the item property page
    Call DisplayUserProps(objitems)

End Sub

Sub DisplayUserProps(ByVal objitems As ItemProperties)
'Displays the names of all user-created item properties in the colle

    For i = 0 To objitems.Count - 1
        'Display name of property if it was created by the user
        If objitems.Item(i).IsUserProperty = True Then
            MsgBox "The property " & objitems(i).Name & " was created"
        End If
    Next i

End Sub
IsVisible Property

Returns a Boolean that determines if the reminder is currently visible. All active reminders are visible. If True, the reminder is visible. Read-only.

expression.IsVisible

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

This property is determined when it is returned based on the state of the current reminder.
Example

The following example dismisses all reminders that are currently visible. For example, if the current reminder is active, the **IsVisible** property will return **True**.

Sub DismissReminders()
'Dismisses any active reminders

    Dim olApp As Outlook.Application
    Dim objRems As Reminders
    Dim objRem As Reminder

    Set olApp = Outlook.Application
    Set objRems = olApp.Reminders

    For Each objRem In objRems
        If objRem.**IsVisible** = True Then
            objRem.Dismiss
        End If
    Next objRem

End Sub
Item Property

Returns an **Object** corresponding to the specified Microsoft Outlook item.

*expression*.**Item**

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

Returns an ItemProperties collection that represents all properties associated with an item.

expression.ItemProperties

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

The **ItemProperties** collection is a zero (0) based collection, meaning that the first object in the collection is referenced by the index 0, instead of 1.
Example

The following example returns the ItemProperties collection associated with a MailItem object.

Sub ItemProperty()
    'Creates a new mail item and accesses its properties

    Dim olApp As Outlook.Application
    Dim objMail As MailItem
    Dim objItems As ItemProperties
    Dim objItem As ItemProperty

    Set olApp = Outlook.Application
    'Create the mail item
    Set objMail = olApp.CreateItem(olMailItem)
    'Create a reference to the item's properties collection
    Set objItems = objMail.ItemProperties
    'Create a reference to the first item property page
    Set objItem = objItems.Item(0)

End Sub
Items Property

- Returns an Items collection as a collection of Outlook items in the specified folder.

  expression.Items

- expression Required. An expression that returns a MAPIFolder object.
Example

This Visual Basic for Applications example uses the **Items** property to obtain the collection of **MailItem** objects from the default Inbox folder.

Set myOlApp = CreateObject("Outlook.Application")
Set myNameSpace = myOlApp.GetNameSpace("MAPI")
Set myFolder = myNameSpace.GetDefaultFolder(olFolderInbox)
Set myItems = myFolder.**Items**

If you use VBScript, you do not create the **Application** object, and you cannot use named constants. This example shows how to perform the same task using VBScript.

Set myNameSpace = Application.GetNameSpace("MAPI")
Set myFolder = myNameSpace.GetDefaultFolder(6)
Set myItems = myFolder.**Items**
JobTitle Property

Returns or sets a String representing the job title for the contact. Read/write.

expression.JobTitle

expression    Required. An expression that returns a ContactItem object.
Journal Property

- True if the transaction of the contact will be journalized. The default value is False. Read/write Boolean.

expression.Journal

expression Required. An expression that returns a ContactItem object.
Language Property

Returns or sets the language setting for the object that defines the language used in the menu. The **Language** property uses a **String** to represent an ISO language tag. For example, the string ""EN-US" represents the ISO code for "United States - English". Read/write.

*expression*.Language

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

If a valid language code is specified, the object will only be available in the View menu for the specified language type. If no value is specified, the object item is available for all language types. The default value for this property is an empty String.
Example

The following example sets the language type of all View object's of type olTableView to US English.

Sub SetLanguage()
'Sets the language of all table views to US English

    Dim olApp As Outlook.Application
    Dim objViews As Views
    Dim objView As View

    Set olApp = Outlook.Application
    Set objViews = olApp.GetNamespace("Mapi").GetDefaultFolder(olFolderInbox).V
    'Iterate through each view in the collection
    For Each objView In objViews
        'If view is of type olTableVView then set language
        If objView.ViewType = olTableView Then
            objView.Language = "EN-US"
        End If
    Next objView

End Sub
LanguageSettings Property

Expression

Returns a LanguageSettings object for the application that contains the language-specific attributes of Microsoft Outlook.

expression-LanguageSettings

expression Required. An expression that returns an Application object.
LastFirstAndSuffix Property

Returns a **String** representing the last name, first name, middle name, and suffix of the contact. There is a comma between the last and first names and spaces between all the names and the suffix. This property is parsed from the **LastName**, **FirstName**, **MiddleName** and **Suffix** properties. Read-only.

**Note** The **LastName**, **FirstName**, **MiddleName**, and **Suffix** properties are themselves parsed from the **FullName** property.

**expression.LastFirstAndSuffix**

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

**Note** The value of this property is only filled when its associated property (**FirstName**, **LastName**, **MiddleName**, **CompanyName**, and **Suffix**) contain Asian (DBCS) characters. If the corresponding field does not contain Asian characters, the property will be empty.
**LastFirstNoSpace Property**

Returns a **String** representing the concatenated last name, first name, and middle name of the contact with no space between the last name and the first name. This property is parsed from the **LastName**, **FirstName** and **MiddleName** properties. Read-only.

**Note** The **LastName**, **FirstName**, and **MiddleName** properties are themselves parsed from the **FullName** property.

**expression**.LastFirstNoSpace

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

**Note** The value of this property is only filled when its associated property (**FirstName**, **LastName**, **MiddleName**, **CompanyName**, and **Suffix**) contain Asian (DBCS) characters. If the corresponding field does not contain Asian characters, the property will be empty.
LastFirstNoSpaceAndSuffix Property

Returns the last name, first name, and suffix of the user without a space. Read-only String.

expression.LastFirstNoSpaceAndSuffix

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

This property is used only when the FirstName, LastName, and Suffix properties (the fields that define this property) contain Asian (DBCS) characters. Note that any such changes or entries to the FirstName, LastName, or Suffix properties will be overwritten by any subsequent changes or entries to FullName.
Returns a String representing the concatenated last name, first name, and middle name of the contact with no space between the last name and the first name. The company name for the contact is included after the middle name. This property is parsed from the LastName, FirstName, MiddleName, and CompanyName properties. Read-only.

**Note** The LastName, FirstName, and MiddleName properties are themselves parsed from the FullName property.

**expression.LastFirstNoSpaceCompany**

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

**Note**  The value of this property is only filled when its associated property (FirstName, LastName, MiddleName, CompanyName, and Suffix) contain Asian (DBCS) characters. If the corresponding field does not contain Asian characters, the property will be empty.
LastFirstSpaceOnly Property

Returns a String representing the concatenated last name, first name, and middle name of the contact with spaces between them. This property is parsed from the LastName, FirstName and MiddleName properties. Read-only.

**Note** The LastName, FirstName, and MiddleName properties are themselves parsed from the FullName property.

```
expression.LastFirstSpaceOnly
```

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

**Note** The value of this property is only filled when its associated property (FirstName, LastName, MiddleName, CompanyName, and Suffix) contain Asian (DBCS) characters. If the corresponding field does not contain Asian characters, the property will be empty.
LastFirstSpaceOnlyCompany Property

Returns a **String** representing the concatenated last name, first name, and middle name of the contact with spaces between them. The company name for the contact is after the middle name. This property is parsed from the **LastName**, **FirstName**, **MiddleName**, and **CompanyName** properties. Read-only.

**Note** The **LastName**, **FirstName**, and **MiddleName** properties are themselves parsed from the **FullName** property.

`expression.LastFirstSpaceOnlyCompany`

`expression`  Required. An expression that returns a **ContactItem** object.

**Note** The value of this property is only filled when its associated property (**FirstName**, **LastName**, **MiddleName**, **CompanyName**, and **Suffix**) contain Asian (DBCS) characters. If the corresponding field does not contain Asian characters, the property will be empty.
LastModificationTime Property

Returns a Date specifying the time that the Microsoft Outlook item was last modified. This property corresponds to the MAPI property PR_LAST_MODIFICATION_TIME. Read-only.

expression.LastModificationTime

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

This Visual Basic for Applications example uses the **Restrict** method to get all Inbox items dealing with Project X and moves them to the Project X folder.

```vba
Set myOlApp = CreateObject("Outlook.Application")
Set myNameSpace = myOlApp.GetNameSpace("MAPI")
Set myFolder = _
    myNameSpace.GetDefaultFolder(olFolderInbox)
Set myItems = myFolder.Items
Set myRestrictItems = myItems.Restrict _
    ("[Categories] = 'Project	X'"
For Each myItem In myRestrictItems
    myItem.Move myFolder.Folders("Project	X")
Next
```

If you use VBScript, you do not create the **Application** object, and you cannot use named constants. This example shows how to perform the same task using VBScript.

```vbs
Set myNameSpace = Application.GetNameSpace("MAPI")
Set myFolder = myNameSpace.GetDefaultFolder(6)
Set myItems = myFolder.Items
Set myRestrictItems = myItems.Restrict _
    ("[Categories] = 'Project	X'"
For Each myItem In myRestrictItems
    myItem.Move myFolder.Folders("Project	X")
Next
```

This Visual Basic for Applications example uses the **Restrict** method to apply a filter to the Contact items based on the item's **LastModificationTime** property.

```vba
Public Sub ContactDateCheck()
    Set myOlApp = CreateObject("Outlook.Application")
    Set myNameSpace = myOlApp.GetNamespace("MAPI")
    Set myContacts = myNameSpace.GetDefaultFolder(olFolderContacts).Items
    Set myItems = myContacts.Restrict("[LastModificationTime] > '05/1"
    For Each myItem In myItems
        MsgBox myItem.FullName & ": " & myItem.LastModificationTime
    Next
End Sub
```

The following Visual Basic for Applications example is the same as the example above, except that it demonstrates the use of a variable in the filter.
Public Sub ContactDateCheck()
    Set myOlApp = CreateObject("Outlook.Application")
    Set myNameSpace = myOlApp.GetNamespace("MAPI")
    Set myContacts = myNameSpace.GetDefaultFolder(olFolderContacts).Items
    DateStart = #6/11/97#
    DateToCheck$ = "[LastModificationTime] >= " & DateStart & ""
    Set myRestrictItems = myContacts.Restrict(DateToCheck$)
    For Each myItem In myRestrictItems
        MsgBox myItem.FullName & ": " & MyItem.LastModificationTime
    Next
End Sub
LastName Property

Returns or sets a String representing the last name for the contact. Read/write.

expression.LastName

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

This property is parsed from the FullName property, but may be changed or entered independently should it be parsed incorrectly. Note that any such changes or entries to this property will be overwritten by any subsequent changes of entries to FullName.
LastNameAndFirstName Property

Returns a String representing the concatenated last name and first name for the contact. Read-only.

expression.LastNameAndFirstName

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

This property is parsed from the **FirstName** and **LastName** properties for the contact, which are themselves parsed from the **FullName** property.
Left Property

- Returns or sets a **Long** specifying the position (in pixels) of the left vertical edge of an explorer, inspector, or note window from the edge of the screen. Read/write.

`expression.Left`

*expression* Required. An expression that returns an **Explorer**, **Inspector**, or **NoteItem** object.
Links Property

- Returns a `Links` collection of `Link` objects that represent the contacts to which the item is linked.

`expression.Links`

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

This Microsoft Visual Basic/Visual Basic for Applications example steps through the items in the Tasks folder and, if a task is not complete, displays the number of contacts linked to the item.

```vba
Dim myOlApp As New Outlook.Application
Dim myNSpace As Outlook.NameSpace
Dim myItems As Outlook.Items
Dim myItem As Outlook.TaskItem
Dim myLinks As Outlook.Links
Dim myLink As Outlook.Link
Set myNSpace = myOlApp.GetNamespace("MAPI")
Set myItems = myNSpace.GetDefaultFolder(olFolderTasks).Items
For x = 1 To myItems.Count
    If TypeName(myItems.Item(x)) = "TaskItem" Then
        Set myItem = myItems.Item(x)
        Set myLinks = myItem.Links
        Msg = myItem.Subject & " has " & myLinks.Count & " links."
        If myItem.Complete = False Then
            If MsgBox(Msg, vbOKCancel) = vbCancel Then Exit For
        End If
    End If
Next x
```

If you use VBScript, you do not create the `Application` object, and you cannot use named constants. This example shows how to perform the same task using VBScript.

```vbs
Set myNSpace = Application.GetNamespace("MAPI")
Set myItems = myNSpace.GetDefaultFolder(13).Items
For x = 1 To myItems.Count
    If TypeName(myItems.Item(x)) = "TaskItem" Then
        Set myItem = myItems.Item(x)
        Set myLinks = myItem.Links
        Msg = myItem.Subject & " has " & myLinks.Count & " links."
        If myItem.Complete = False Then
            If MsgBox(Msg, 1) = 2 Then Exit For
        End If
    End If
Next x
```
Location Property

Returns or sets a **String** representing the specific office location (for example, Building 1 Room 1 or Suite 123) for the appointment. This property corresponds to the MAPI property PR_OFFICE_LOCATION. Read/write.

*expression*.Location

*expression*  Required. An expression that returns an [AppointmentItem](#) object.
Example

This Visual Basic for Applications example uses **CreateItem** to create an appointment and uses **MeetingStatus** to set the meeting status to "Meeting" to turn it into a meeting request with both a required and an optional attendee.

Set myOlApp = CreateObject("Outlook.Application")
Set myItem = myOlApp.CreateItem(olAppointmentItem)
myItem.MeetingStatus = olMeeting
myItem.Subject = "Strategy Meeting"
myItem.Location = "Conference Room B"
myItem.Start = #9/24/97 1:30:00 PM#
myItem.Duration = 90
Set myRequiredAttendee = myItem.Recipients.Add ("Nate Sun")
myRequiredAttendee.Type = olRequired
Set myOptionalAttendee = myItem.Recipients.Add ("Kevin Kennedy")
myOptionalAttendee.Type = olOptional
Set myResourceAttendee = myItem.Recipients.Add("Conference Room B")
myResourceAttendee.Type = olResource
myItem.Send

If you use VBScript, you do not create the **Application** object, and you cannot use named constants. This example shows how to perform the same task using VBScript.

Set myItem = Application.CreateItem(1)
myItem.MeetingStatus = 1
myItem.Subject = "Strategy Meeting"
myItem.Location = "Conference Room B"
myItem.Start = #9/24/97 1:30:00 PM#
myItem.Duration = 90
Set myRequiredAttendee = myItem.Recipients.Add ("Nate Sun")
myRequiredAttendee.Type = 1
Set myOptionalAttendee = myItem.Recipients.Add ("Kevin Kennedy")
myOptionalAttendee.Type = 2
Set myResourceAttendee = myItem.Recipients.Add("Conference Room B")
myResourceAttendee.Type = 3
myItem.Send
Locked Property

True if the form cannot be modified. Read/write Boolean.

expression.Locked

expression Required. An expression that returns a FormDescription object.
LockUserChanges Property

Returns or sets a value that indicates whether a user can modify the settings of the current view. Read/write Boolean.

expression.LockUserChanges

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

If True, the user can modify the settings of the current view; however, changes made to the interface will not be saved. If False (the default), any changes will be saved.
Example

The following example locks the user interface for all views that are available to all users. The subroutine LockView accepts the View object and a Boolean value that indicates if the View interface will be locked. In this example the procedure is always called with the Boolean value set to True.

Sub LocksPublicViews()
'Locks the interface of all views that are available to 'all users of this folder

    Dim olApp As Outlook.Application
    Dim objName As NameSpace
    Dim objViews As Views
    Dim objView As View

    Set olApp = Outlook.Application
    Set objName = olApp.GetNamespace("MAPI")
    Set objViews = objName.GetDefaultFolder(olFolderNotes).Views

    For Each objView In objViews
        If objView.SaveOption = olViewSaveOptionThisFolderEveryone Then
            Call LockView(objView, True)
        End If
    Next objView

End Sub

Sub LockView(ByRef objView As View, ByVal blnAns As Boolean)
'locks the user interface of the view
'accepts and returns a View object and user response

    With objView
        If blnAns = True Then
            'if true lock UI
            .LockUserChanges = True
            .Save
        Else
            'if false don't lock UI
            .LockUserChanges = False
        End If
    End With

End Sub
MailingAddress Property

- Returns or sets a String representing the full, unparsed selected mailing address for the contact. Read/write.

  expression.MailingAddress

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

This property replicates the property indicated by the SelectedMailingAddress property, which is one of the following OlMailingAddress constants: olBusiness, olHome, olNone, or olOther. While it can be changed or entered independently, any such changes or entries to this property will be overwritten by any subsequent changes or entries to the property indicated by SelectedMailingAddress.
MailingAddressCity Property

Returns or sets a String representing the city name portion of the selected mailing address of the contact. Read/write.

expression.MailingAddressCity

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

This property replicates the property indicated by the `SelectedMailingAddress` property, which is one of the following `OlMailingAddress` constants: `olBusiness`, `olHome`, `olNone`, or `olOther`. While it can be changed or entered independently, any such changes or entries to this property will be overwritten by any subsequent changes or entries to the property indicated by `SelectedMailingAddress`. 
MailingAddressCountry Property

Returns or sets a **String** representing the country/region code portion of the selected mailing address of the contact. Read/write.

`expression.MailingAddressCountry`

`expression` Required. An expression that returns a **ContactItem** object.
Remarks

This property replicates the property indicated by the `SelectedMailingAddress` property, which is one of the following `OlMailingAddress` constants: `olBusiness`, `olHome`, `olNone`, or `olOther`. While it can be changed or entered independently, any such changes or entries to this property will be overwritten by any subsequent changes or entries to the property indicated by `SelectedMailingAddress`. 
MailingAddressPostalCode Property

Returns or sets a String representing the postal code (zip code) portion of the selected mailing address of the contact. Read/write.

expression.MailingAddressPostalCode

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

This property replicates the property indicated by the SelectedMailingAddress property, which is one of the following OlMailingAddress constants: olBusiness, olHome, olNone, or olOther. While it can be changed or entered independently, any such changes or entries to this property will be overwritten by any subsequent changes or entries to the property indicated by SelectedMailingAddress.
MailingAddressPostOfficeBox Property

Returns or sets a String representing the post office box number portion of the selected mailing address of the contact. Read/write.

expression.MailingAddressPostOfficeBox

description Required. An expression that returns a ContactItem object.
Remarks

This property replicates the property indicated by the `SelectedMailingAddress` property, which is one of the following `OlMailingAddress` constants: `olBusiness`, `olHome`, `olNone`, or `olOther`. While it can be changed or entered independently, any such changes or entries to this property will be overwritten by any subsequent changes or entries to the property indicated by `SelectedMailingAddress`. 
MailingAddressState Property

- Returns or sets a **String** representing the state code portion for the selected mailing address of the contact. Read/write.

```
expression.MailingAddressState
```

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

This property replicates the property indicated by the `SelectedMailingAddress` property, which is one of the following `OlMailingAddress` constants: `olBusiness`, `olHome`, `olNone`, or `olOther`. While it can be changed or entered independently, any such changes or entries to this property will be overwritten by any subsequent changes or entries to the property indicated by `SelectedMailingAddress`. 
MailingAddressStreet Property

Returns or sets a String representing the street address portion of the selected mailing address of the contact. Read/write.

expression.MailingAddressStreet

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

This property replicates the property indicated by the `SelectedMailingAddress` property, which is one of the following `OlMailingAddress` constants: `olBusiness`, `olHome`, `olNone`, or `olOther`. While it can be changed or entered independently, any such changes or entries to this property will be overwritten by any subsequent changes or entries to the property indicated by `SelectedMailingAddress`. 
Manager Property

- Returns an **AddressEntry** object that represents the manager of the user that corresponds to this address entry. If the user's manager is not available in the messaging system, the **Manager** property returns **Nothing**.

  `expression.Manager`

  `expression`  Required. An expression that returns an **AddressEntry** object.
ManagerName Property

Returns or sets a String representing the manager name for the contact. Read/write.

expression.ManagerName

expression  Required. An expression that returns a ContactItem object.
MarkForDownload Property

Returns or sets an OlRemoteStatus constant that determines the status of an item once it is received by a remote user. This property gives remote users with less-than-ideal data-transfer capabilities increased messaging flexibility. Read/write.

OlRemoteStatus can be one of these OlRemoteStatus constants.

- **olMarkedForCopy** The item will be copied to the remote site.
- **olMarkedForDelete** The item will be deleted.
- **olMarkedForDownload** The item will be downloaded in its entirety.
- **olRemoteStatusNone** The item has no remote status.
- **olUnMarked** The item isn't marked for remote status and will be disregarded.

expression.**MarkForDownload**

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

The following subroutine sets the **MarkForDownload** property of all e-mail items sent by a specified user to **olMarkedForDownload**. All items from the specified user will then be downloaded in their entirety.

Sub RemoteUse(ByVal strSender As String, ByRef objMail As MailItem)
' Marks all items from a certain user to be downloaded

    With objMail
        ' If mail is from the sender, mark it for download
        If .SenderName = strSender Then
            .MarkForDownload = olMarkedForDownload
        End If
    End With

End Sub
MeetingResponseStatus Property

- Returns an OlResponseStatus constant indicating the overall status of the response to the meeting request for the recipient. Read-only.

OlResponseStatus can be one of these OlResponseStatus constants.

- olResponseAccepted
- olResponseDeclined
- olResponseNone
- olResponseNotResponded
- olResponseOrganized
- olResponseTentative

expression.MeetingResponseStatus

expression  Required. An expression that returns a Recipient object.
MeetingStatus Property

- Returns or sets an OlMeetingStatus constant specifying the meeting status of the appointment. Use this property to make a MeetingItem object available for the appointment. Read/write.

OlMeetingStatus can be one of these OlMeetingStatus constants.

- olMeeting
- olMeetingCanceled
- olMeetingReceived
- olNonMeeting

expression.MeetingStatus

expression Required. An expression that returns an AppointmentItem object.
**Example**

This Visual Basic for Applications example uses `CreateItem` to create an appointment and uses `MeetingStatus` to set the meeting status to "Meeting" to turn it into a meeting request with both a required and an optional attendee.

```vbnet
Set myOlApp = CreateObject("Outlook.Application")
Set myItem = myOlApp.CreateItem(olAppointmentItem)
myItem.MeetingStatus = olMeeting
myItem.Subject = "Strategy Meeting"
myItem.Location = "Conference Room B"
myItem.Start = #9/24/1997 1:30:00 PM#
myItem.Duration = 90
Set myRequiredAttendee = myItem.Recipients.Add("Nate Sun")
myRequiredAttendee.Type = olRequired
Set myOptionalAttendee = myItem.Recipients.Add("Kevin Kennedy")
myOptionalAttendee.Type = olOptional
Set myResourceAttendee = myItem.Recipients.Add("Conference Room B")
myResourceAttendee.Type = olResource
myItem.Display
```

If you use VBScript, you do not create the `Application` object, and you cannot use named constants. This example shows how to perform the same task using VBScript.

```vbnet
Set myItem = Application.CreateItem(1)
myItem.MeetingStatus = 1
myItem.Subject = "Strategy Meeting"
myItem.Location = "Conference Room B"
myItem.Start = #9/24/97 1:30:00 PM#
myItem.Duration = 90
Set myRequiredAttendee = myItem.Recipients.Add("Nate Sun")
myRequiredAttendee.Type = 1
Set myOptionalAttendee = myItem.Recipients.Add("Kevin Kennedy")
myOptionalAttendee.Type = 2
Set myResourceAttendee = myItem.Recipients.Add("Conference Room B")
myResourceAttendee.Type = 3
myItem.Display
```
MemberCount Property

Returns a **Long** indicating the number of members in a distribution list. Read-only.

*expression*.MemberCount

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

The value returned represents all members of the distribution list, including member distribution lists. Each member distribution list is counted as a single member. That is, **MemberCount** is not an aggregate sum of the recipients in the distribution list plus recipients in member distribution lists. For example, if a distribution list contains 10 recipients plus one distribution list containing 15 recipients, **MemberCount** returns 11.
Example

This Microsoft Visual Basic/Visual Basic for Applications example steps through the default Contacts folder, and if it finds a distribution list with more than 20 members it displays the item.

Dim myOlApp As New Outlook.Application
Dim myOlFolder As Outlook.MAPIFolder
Dim myOlItems As Outlook.Items
Dim myOlDistList As Outlook.DistListItem
Set myOlFolder = myOlApp.GetNamespace("MAPI").GetDefaultFolder(olFolderContacts)
Set myOlItems = myOlFolder.Items
For x = 1 To myOlItems.Count
    If TypeName(myOlItems.Item(x)) = "DistListItem" Then
        Set myOlDistList = myOlItems.Item(x)
        If myOlDistList.MemberCount > 20 Then
            MsgBox myOlDistList.DLName & " has more than 20 members."
            myOlDistList.Display
        End If
    End If
Next x

If you use VBScript, you do not create the Application object, and you cannot use named constants. This example shows how to perform the same task using VBScript.

Set myOlFolder = _
    Application.GetNamespace("MAPI").GetDefaultFolder(10)
Set myOlItems = myOlFolder.Items
For x = 1 To myOlItems.Count
    If TypeName(myOlItems.Item(x)) = "DistListItem" Then
        Set myOlDistList = myOlItems.Item(x)
        If myOlDistList.MemberCount > 20 Then
            MsgBox myOlDistList.DLName & _
                " has more than 20 members."
            myOlDistList.Display
        End If
    End If
Next x
**Members Property**

Returns an AddressEntries collection object (for the AddressEntries object) or a Variant (for the DistListItem object) representing the members (for the AddressEntries object) or a member (for the DistListItem object) of a distribution list in an address book.

A distribution list is an AddressEntry object whose DisplayType property is set to olDistList or olPrivateDistList.

The Members property returns Nothing if the AddressEntry is not a distribution list.

Read-only for the AddressEntries object; read/write for the DistListItem object.

`expression.Members`

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

Returns or sets a **String** representing the message class for the Microsoft Outlook item or **Action**. This property corresponds to the MAPI property PR_MESSAGE_CLASS. The **MessageClass** property links the **item** to the form on which it is based. When an item is selected, Outlook uses the message class to locate the form and expose its properties, such as Reply commands. Read-only for the **FormDescription** object; read/write for all other objects in the Applies To list.

`expression.MessageClass`

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

Returns or sets a String representing the middle name for the contact. Read/write.

\[ expression\text{.MiddleName} \]

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

This property is parsed from the **FullName** property, but may be changed or entered independently should it be parsed incorrectly. Note that any such changes or entries to this property will be overwritten by any subsequent changes of entries to **FullName**.
Mileage Property

Returns or sets a **String** representing the mileage for an **item**. This is a free-form string field and can be used to store mileage information associated with the item (for example, 100 miles documented for an appointment, contact, or task) for purposes of reimbursement. Read/write.

*expression*.Mileage

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

- Returns or sets a `String` representing the file name of the mini-icon to be displayed for the form. Read/write.

`expression.MiniIcon`

`expression` Required. An expression that returns a `FormDescription` object.
MobileTelephoneNumber Property

Returns or sets a String representing the mobile telephone number for the contact. Read/write.

expression.MobileTelephoneNumber

expression Required. An expression that returns a ContactItem object.
ModifiedFormPages Property

Returns the **Pages** collection that represents all the pages for the **item** in the inspector. The main page and up to five customizable pages can be obtained using the **Add** method.

*expression*.ModifiedFormPages

*expression* Required. An expression that returns an **Inspector** object.
Example

This Visual Basic for Applications example returns a new custom page named "My Page" (the same call without a name will return the next available page for customization — for example, "Custom1").

Set myOlApp = CreateObject("Outlook.Application")
Set myItem = myOlApp.CreateItem(olContactItem)
Set myPages = myItem.GetInspector.ModifiedFormPages
Set myPage = myPages.Add("My Page")
MonthOfYear Property

Returns or sets a Long indicating which month of the year is valid for the specified recurrence pattern. Can be a number from 1 through 12. For example, setting this property to 5 and the RecurrenceType property to olRecursYearly would cause this recurrence pattern to occur every May. Read/write.

expression.MonthOfYear

desired-expression Required. An expression that returns a RecurrencePattern object.

Note This property is only valid for recurrence patterns whose RecurrenceType property is set to olRecursYearly or olRecursYearNth.
Name Property

Returns or sets the display name for the item. The Name property is also the caption for a form. Read/write depending on the object.

expression.Name

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

Note The Name property must be set before you can use the PublishForm method. It is also necessary for the Name property to be set before calling the Details method.
Example

This Visual Basic for Applications example uses the `CurrentUser` property to obtain the name of the currently logged-on user and then displays a message box containing the name.

```vba
Set myOlApp = CreateObject("Outlook.Application")
Set myNameSpace = myOlApp.GetNameSpace("MAPI")
MsgBox myNameSpace.CurrentUser.Name
```

If you use VBScript, you do not create the `Application` object. This example shows how to perform the same task using VBScript.

```vbs
Set myNameSpace = Application.GetNameSpace("MAPI")
MsgBox myNameSpace.CurrentUser.Name
```
NetMeetingAlias Property

Returns or sets a String indicating the user's Microsoft NetMeeting ID, or alias. Read/write.

expression.NetMeetingAlias

expression  Required. An expression that returns a ContactItem object.
**NetMeetingAutoStart Property**

True if this online meeting starts automatically. Read/write **Boolean**.

`expression.NetMeetingAutoStart`  

*expression*  Required. An expression that returns an **AppointmentItem** object.
NetMeetingDocPathName Property

Returns or sets a String representing the full path to the Microsoft Office document specified for a Microsoft NetMeeting online meeting. Read/write.

expression.NetMeetingDocPathName

type expression Required. An expression that returns an AppointmentItem object.
NetMeetingOrganizerAlias Property

Returns or sets a String representing the alias of the meeting organizer, if this is an online meeting. Read/write.

expression.NetMeetingOrganizerAlias

expression Required. An expression that returns an AppointmentItem object.
NetMeetingServer Property

Returns or sets a String specifying the name of the Microsoft NetMeeting server being used for an online meeting. Read/write.

expression.NetMeetingServer

description Required. An expression that returns an AppointmentItem or ContactItem object.
NetMeetingType Property

Sets or returns an **OlNetMeetingType** constant specifying the type of Microsoft NetMeeting. Read/write.

OlNetMeetingType can be one of these OlNetMeetingType constants.

- **olChat**
- **olNetMeeting**
- **olNetShow**

*expression*.NetMeetingType

*expression*  Required. An expression that returns an [AppointmentItem](#) object.
NetShowURL Property

Returns or sets a **String** specifying the URL for a Microsoft NetShow online meeting. Read/write.

`expression.NetShowURL`

`expression` Required. An expression that returns an **AppointmentItem** object.
NextReminderDate Property

Returns a Date that indicates the next time the specified reminder will occur. Read-only.

expression.NextReminderDate

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

The **NextReminderDate** property value changes every time the object's **Snooze** method is executed or when the user clicks the **Snooze** button.
Example

The following example creates a report of all reminders in the collection and the dates when they will next occur. The subroutine concatenates the Caption and NextReminderDate properties into a string and displays the string in a dialog box.

Sub DisplayNextDateReport()
' Displays the next time all reminders will be displayed

    Dim olApp As Outlook.Application
    Dim objRems As Reminders
    Dim objRem As Reminder
    Dim strTitle As String
    Dim strReport As String

    Set olApp = Outlook.Application
    Set objRems = olApp.Reminders

    strTitle = "Current Reminder Schedule:
    ' Check if any reminders exist
    If objRems.Count = 0 Then
        MsgBox "There are no current reminders."
    Else
        For Each objRem In objRems
            ' If string is empty, create new string
            If strReport = "" Then
                strReport = objRem.Caption & vbCrLf & _
                objRem.NextReminderDate & vbCrLf
            Else
                ' Add info to string
                objRem.NextReminderDate & vbCrLf
            End If
            Next objRem
        ' Display report in dialog
        MsgBox strTitle & vbCrLf & vbCrLf & strReport
    End If
End Sub
NickName Property

- Returns or sets a String representing the nickname for the contact. Read/write.

expression.Nickname

expression Required. An expression that returns a ContactItem object.
NoAging Property

- True to not age the Outlook item. Read/write Boolean.

expression.NoAging

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

- **True** if the recurrence pattern has no end date. Read/write **Boolean**.

*expression*.NoEndDate

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

This property must be coordinated with other properties when setting up a recurrence pattern. If the \texttt{PatternEndDateTime} property or the \texttt{Occurrences} property is set, the pattern is considered to be finite and the \texttt{NoEndDateTime} property is \texttt{False}. If neither \texttt{PatternEndDateTime} nor \texttt{Occurrences} is set, the pattern is considered infinite and \texttt{NoEndDateTime} is \texttt{True}. 
Number Property

Returns or sets a String corresponding to the number for the specified form. Read/write.

`expression.Number`

`expression` Required. An expression that returns a `FormDescription` object.
Occurrences Property

Returns or sets a `Long` indicating the number of occurrences of the recurrence pattern. This property allows the definition of a recurrence pattern that is only valid for the specified number of subsequent occurrences. For example, you can set this property to 10 for a formal training course that will be held on the next ten Thursday evenings. Read/write.

`expression.Occurrences`

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

This property must be coordinated with other properties when setting up a recurrence pattern. If the PatternEndDate property or the Occurrences property is set, the pattern is considered to be finite and the NoEndDate property is False. If neither PatternEndDate nor Occurrences is set, the pattern is considered infinite and NoEndDate is True.
OfficeLocation Property

Returns or sets a **String** specifying the specific office location (for example, Building 1 Room 1 or Suite 123) for the contact. This property corresponds to the MAPI property PR_OFFICE_LOCATION. Read/write.

*expression*.OfficeLocation

*expression* Required. An expression that returns a **ContactItem** object.
Offline Property

Returns a **Boolean** indicating whether Microsoft Outlook is online (connected to a server) or Offline (not connected to a server). Returns **True** if Outlook is offline. Read-only.

*expression*.Offline

*expression*  Required. An expression that returns a **Namespace** object.
Example

The following function returns **True** or **False** depending on whether the Namespace object is currently online.

Sub Off()
' Determines whether Outlook is currently Offline
    Dim olapp As New Outlook.Application
    Dim nmsName As NameSpace
    Set olapp = Outlook.Application
    Set nmsName = olapp.GetNamespace("MAPI")
    Dim blnOff As Boolean
    blnOff = IsOffline(nmsName)
End Sub

Function IsOffline(ByVal nmsName As NameSpace) As Boolean
' Returns True if Outlook is offline
    IsOffline = nmsName.offline
End Function
OneOff Property

True if the form will be discarded after using once (one-off). False if the form is retained as a custom form. Read/write Boolean.

expression.ONEOFF

expression Required. An expression that returns a FormDescription object.
OptionalAttendees Property

Returns or sets a String representing the display string of optional attendees names for the appointment. This property corresponds to the MAPI property PR_DISPLAY_CC. Read/write.

expression.OptionalAttendees

expression Required. An expression that returns an AppointmentItem object.
Ordinal Property

Returns or sets a `Long` specifying the position in the view (ordinal) for the task. Read/write.

`expression.Ordinal`

`expression`  Required. An expression that returns a `TaskItem` object.
OrganizationalIDNumber Property

Returns or sets a String representing the organizational ID number for the contact. Read/write.

expression.OrganizationalIDNumber

expression Required. An expression that returns a ContactItem object.
Organizer Property

Returns a **String** representing the name of the organizer of the appointment. Read-only.

*expression.Organizer*

*expression* Required. An expression that returns an **AppointmentItem** object.
OriginalDate Property

Returns a Date indicating the original date and time of anAppointmentItem before it was altered. This property will return the original date even if theAppointmentItem has been deleted but it will not return the original time if deletion has occurred. Read-only.

expression.OriginalDate

expression  Required. An expression that returns an Exception object.
Example

This Visual Basic for Applications example uses CreateItem to create an AppointmentItem object. The RecurrencePattern is obtained for this item using the GetRecurrencePattern method. By setting the RecurrencePattern properties, RecurrenceType, PatternStartDate, and PatternEndDate, the appointments are now a recurring series that occur on a daily basis for the period of one year.

An Exception object is created when one instance of this recurring appointment is obtained using the GetOccurrence method and properties for this instance are altered. This exception to the series of appointments is obtained using the GetRecurrencePattern method to access the Exceptions collection associated with this series. Message boxes display the original Subject and OriginalDate for this exception to the series of appointments and the current date, time, and subject for this exception.

For a description of changes required for this example to work in VBScript, see the Note at the end of the example.

Public Sub cmdExample()
    Set myOlApp = New Outlook.Application
    Set myApptItem = myOlApp.CreateItem(olAppointmentItem)
    myApptItem.Start = #2/2/98 3:00:00 PM#
    myApptItem.End = #2/2/98 4:00:00 PM#
    myApptItem.Subject = "Meet with Boss"

    'Get the recurrence pattern for this appointment
    'and set it so that this is a daily appointment
    'that begins on 2/2/98 and ends on 2/2/99
    'and save it.
    Set myRecurrPatt = myApptItem.GetRecurrencePattern
    myRecurrPatt.RecurrenceType = olRecursDaily
    myRecurrPatt.PatternStartDate = #2/2/98#
    myRecurrPatt.PatternEndDate = #2/2/99#
    myApptItem.Save

    'Access the items in the Calendar folder to locate
    'the master AppointmentItem for the new series.
    Set myNamespace = myOlApp.GetNamespace("MAPI")
    Set myFolder = myNamespace.GetDefaultFolder(olFolderCalendar)
Set myItems = myFolder.Items
Set myApptItem = myItems("Meet with Boss")

'Get the recurrence pattern for this appointment
'and obtain the occurrence for 3/12/98.
myDate = #3/12/98 3:00:00 PM#
Set myRecurrPatt = myApptItem.GetRecurrencePattern
Set myOddApptItem = myRecurrPatt.GetOccurrence(myDate)

'Save the existing subject. Change the subject and
'starting time for this particular appointment
'and save it.
saveSubject = myOddApptItem.Subject
myOddApptItem.Subject = "Meet NEW Boss"
newDate = #3/12/98 3:30:00 PM#
myOddApptItem.Start = newDate
myOddApptItem.Save

'Get the recurrence pattern for the master
'AppointmentItem. Access the collection of
'exceptions to the regular appointments.
Set myRecurrPatt = myApptItem.GetRecurrencePattern
Set myException = myRecurrPatt.Exceptions.Item(1)

'Display the original date, time, and subject
'for this exception.
MsgBox myException.OriginalDate & ": " & saveSubject

'Display the current date, time, and subject
'for this exception.
MsgBox myException.AppointmentItem.Start & ": " & _
myException.AppointmentItem.Subject
End Sub

Note For this example to work properly in VBScript, there are only a few simple changes that need to be made in the code.

You don't have to retrieve the application as an object and you must use the values of the constants, so:

Set myOlApp = New Outlook.Application
        Set myApptItem = myOlApp.CreateItem(olAppointmentItem)

becomes:

Set myApptItem = Application.CreateItem(1)
and

myRecurrPatt.RecurrenceType = olRecurrsDaily

becomes:

myRecurrPatt.RecurrenceType = 0

and

Set myFolder = myNamespace.GetDefaultFolder(olFolderCalendar)

becomes:

Set myFolder = myNamespace.GetDefaultFolder(9)
OriginalReminderDate Property

Returns a Date that specifies the original date and time that the specified reminder is set to occur. Read-only.

expression.OriginalReminderDate

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

This value corresponds to the original date and time value before the Snooze method is executed or the user clicks the Snooze button.
Example

The following example creates a report of all reminders in the Reminders collection and the dates at which they are scheduled to occur. The subroutine concatenates the Caption and OriginalReminderDate properties of all Reminder objects in the collection into a string and displays the string in a dialog box.

Sub DisplayOriginalDateReport()
'Displays the time at which all reminders will be displayed

    Dim olApp As Outlook.Application
    Dim objRems As Reminders
    Dim objRem As Reminder
    Dim strTitle As String
    Dim strReport As String

    Set olApp = Outlook.Application
    Set objRems = olApp.Reminders

    strTitle = "Original Reminder Schedule:")
    'Check if any reminders exist
    If objRems.Count = 0 Then
        MsgBox "There are no current reminders."
    Else
        For Each objRem In objRems
            'If string is empty, create new string
            If strReport = "" Then
                strReport = objRem.Caption & vbCrLf & vbCrLf & _
                objRem.OriginalReminderDate & vbCrLf
            Else
                'Add info to string
                strReport = strReport & objRem.Caption & vbCrLf & vbCrLf & _
                objRem.OriginalReminderDate & vbCrLf
            End If
        Next objRem
        'Display report in dialog
        MsgBox strTitle & vbCrLf & vbCrLf & strReport
    End If

End Sub
OriginatorDeliveryReportRequested Property

Returns or sets a **Boolean** value that determines whether the originator of the meeting item or mail message will receive a delivery report. Each transport provider that handles your message sends you a single delivery notification containing the names and addresses of each recipient to whom it was delivered. Note that delivery does not imply that the message has been read. The **OriginatorDeliveryReportRequested** property corresponds to the MAPI property PR_ORIGINATOR_DELIVERY_REPORT_REQUESTED. **True** if the originator requested a delivery receipt on the message. Read/write.

**expression.OriginatorDeliveryReportRequested**

**expression** Required. An expression that returns a **MailItem** or a **MeetingItem** object.
OtherAddress Property

- Returns or sets a **String** representing the other address for the contact. Read/write.

```
expression. OtherAddress
```

`expression` Required. An expression that returns a **ContactItem** object.
Remarks

This property contains the full, unparsed other address for the contact.
OtherAddressCity Property

- Returns or sets a String representing the city portion of the other address for the contact. Read/write.

expression.OtherAddressCity

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

This property is parsed from the OtherAddress property, but may be changed or entered independently should it be parsed incorrectly. Note that any such changes or entries to this property will be overwritten by any subsequent changes or entries to OtherAddress.
OtherAddressCountry Property

Returns or sets a **String** representing the country/region portion of the other address for the contact. Read/write.

*expression*.OtherAddressCountry

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

This property is parsed from the OtherAddress property, but may be changed or entered independently should it be parsed incorrectly. Note that any such changes or entries to this property will be overwritten by any subsequent changes or entries to OtherAddress.
OtherAddressPostalCode Property

Returns or sets a **String** representing the postal code portion of the other address for the contact. Read/write.

*expression*.OtherAddressPostalCode

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

This property is parsed from the OtherAddress property, but may be changed or entered independently should it be parsed incorrectly. Note that any such changes or entries to this property will be overwritten by any subsequent changes or entries to OtherAddress.
OtherAddressPostOfficeBox Property

Returns or sets a String representing the post office box portion of the other address for the contact. Read/write.

expression.OtherAddressPostOfficeBox

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

This property is parsed from the OtherAddress property, but may be changed or entered independently should it be parsed incorrectly. Note that any such changes or entries to this property will be overwritten by any subsequent changes or entries to OtherAddress.
OtherAddressState Property

- Returns or sets a String representing the state portion of the other address for the contact. Read/write.

\[expression.\text{OtherAddressState}\]

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

This property is parsed from the OtherAddress property, but may be changed or entered independently should it be parsed incorrectly. Note that any such changes or entries to this property will be overwritten by any subsequent changes or entries to OtherAddress.
OtherAddressStreet Property

- Returns or sets a String representing the street portion of the other address for the contact. Read/write.

expression.OtherAddressStreet

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

This property is parsed from the OtherAddress property, but may be changed or entered independently should it be parsed incorrectly. Note that any such changes or entries to this property will be overwritten by any subsequent changes or entries to OtherAddress.
OtherFaxNumber Property

- Returns or sets a **String** representing the other fax number for the contact. Read/write.

  `expression.OtherFaxNumber`

  `expression` Required. An expression that returns a **ContactItem** object.
OtherTelephoneNumber Property

Returns or sets a **String** representing the other telephone number for the contact. Read/write.

`expression.OtherTelephoneNumber`

`expression` Required. An expression that returns a **ContactItem** object.
OutlookInternalVersion Property

Returns a Long representing the build number of the Microsoft Outlook application for an Outlook item. Read-only.

expression.OutlookInternalVersion

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

Returns a String indicating the major and minor version number of the Microsoft Outlook application for an Outlook item. Read-only.

expression.OutletVersion

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

Returns or sets a **String** indicating the owner for the task. This is a free-form string field. Setting this property to someone other than the current user does not have the effect of delegating the task. Read/write.

*expression*.Owner

*expression*  Required. An expression that returns a **TaskItem** object.
Ownership Property

Returns an OlTaskOwnership specifying the ownership state of the task. Read-only.

OlTaskOwnership can be one of these OlTaskOwnership constants.

olDelegatedTask
olNewTask
olOwnTask

expression.Ownership

expression Required. An expression that returns a TaskItem object.
Returns or sets a String representing the pager number for the contact. Read/write.

```
expression.PagerNumber
```

expression Required. An expression that returns a ContactItem object.
**Panes Property**

Returns a **Panes** collection object representing the panes displayed by the specified explorer.

*expression*.Panes

*expression*  Required. An expression that returns an **Explorer** object.
Example

This Microsoft Visual Basic/Visual Basic for Applications example adds a group named Marketing as the second group in the Outlook Bar.

```vba
Dim myOlApp As New Outlook.Application
Dim myolBar As Outlook.OutlookBarPane
Set myolBar = myOlApp.ActiveExplorer.Panes.Item("OutlookBar")
```

If you use VBScript, you do not create the `Application` object. This example shows how to perform the same task using VBScript.

```vbnet
Set myolBar = Application.ActiveExplorer.Panes.Item("OutlookBar")
```
Parent Property

Returns the parent **Object** of the specified object. Read-only.

`expression.Parent`

`expression` Required. An expression that returns an Outlook object.
Example

This VBScript example uses the Reply event and sets the Sent Items folder for the reply item to the folder in which the original item resides.

Function Item_Reply(ByVal myResponse)
    Set myResponse.SaveSentMessageFolder = Item.Parent
End Function
Password Property

- Returns or sets a String specifying the password for modifying the form. Read/write.

expression.Password

expression Required. An expression that returns a FormDescription object.
PathName Property

Returns a **String** representing the full path to the linked attached file. This property is only valid for linked files. Read-only.

`expression.PathName`

*expression* Required. An expression that returns an **Attachment** object.
PatternEndDate Property

- Returns or sets a Date indicating the end date for the recurrence pattern. Read/write.

expression.PatternEndDate

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

This property is optional but must be coordinated with other properties when setting up a recurrence pattern. If this property or the **Occurrences** property is set, the pattern is considered to be finite, and the **NoEndDate** property is **False**. If neither **PatternEndDate** nor **Occurrences** is set, the pattern is considered infinite and **NoEndDate** is **True**. The **Interval** property must be set before setting **PatternEndDate**.
PatternStartDate Property

Returns or sets a Date indicating the start date for the recurrence pattern. Read/write.

expression.PatternStartDate

expression Required. An expression that returns a RecurrencePattern object.
PercentComplete Property

- Returns or sets a Long indicating the percentage of the task completed at the current date and time. Read/write.

expression.PercentComplete

expression Required. An expression that returns a TaskItem object.
PersonalHomePage Property

- Returns or sets a **String** representing the URL of the personal Web page for the contact. Read/write.

*expression*.PersonalHomePage

*expression* Required. An expression that returns a **ContactItem** object.
Position Property

Returns or sets a **Long** indicating the position of the attachment within the body of the **item**. Read/write.

*expression*.Position

*expression* Required. An expression that returns an **Attachment** object.
Prefix Property

Returns or sets a String specifying the prefix (for example, "Re") to use with the subject of the item when the action is executed. Note that Outlook automatically adds a colon (:) to the value of the Prefix property when setting the subject of the item. Read/write.

expression.Prefix

expression Required. An expression that returns an Action object.
PrimaryTelephoneNumber Property

- Returns or sets a String specifying the primary telephone number for the contact. Read/write.

\[expression.\text{PrimaryTelephoneNumber}\]

expression Required. An expression that returns a ContactItem object.
ProductCode Property

Returns a String specifying the Microsoft Outlook globally unique identifier (GUID). Read-only.

expression.ProductCode()

expression Required. An expression that returns an Application object.
Profession Property

- Returns or sets a **String** indicating the profession for the contact. Read/write.

  *expression*.Profession

*expression*  Required. An expression that returns a **ContactItem** object.
**RadioTelephoneNumber Property**

Returns or sets a `String` indicating the radio telephone number for the contact. Read/write.

`expression.RadioTelephoneNumber`

`expression` Required. An expression that returns a `ContactItem` object.
ReadReceiptRequested Property

True if a read receipt has been requested by the sender. This property corresponds to the MAPI property PR_READ_RECEIPT_REQUESTED. Read/write.

expression.ReadReceiptRequested

expression  Required. An expression that returns a MailItem object.
**ReceivedByEntryID Property**

Returns a `String` representing the `EntryID` for the true recipient as set by the transport provider delivering the mail message. This property corresponds to the MAPI property PR_RECEIVED_BY_ENTRYID. Read-only.

*expression*.ReceivedByEntryID

*expression*  Required. An expression that returns a `MailItem` object.
**ReceivedByName Property**

Returns a `String` representing the display name of the true recipient for the mail message. This property corresponds to the MAPI property PR_RECEIVED_BY_NAME. Read-only.

`expression.ReceivedByName`

`expression`  Required. An expression that returns a `MailItem` object.
**ReceivedOnBehalfOfEntryID Property**

Returns a `String` representing the `EntryID` of the user delegated to represent the recipient for the mail message. This property corresponds to the MAPI property `PR_RCVD_REPRESENTING_ENTRYID`. Read-only.

`expression.ReceivedOnBehalfOfEntryID`

`expression`  Required. An expression that returns a `MailItem` object.
ReceivedOnBehalfOfName Property

Returns a String representing the display name of the user delegated to represent the recipient for the mail message. This property corresponds to the MAPI property PR_RCVD_REPRESENTING_NAME. Read-only.

expression.ReceivedOnBehalfOfName

expression Required. An expression that returns a MailItem object.
ReceivedTime Property

Returns or sets a Date indicating the date and time at which the mail message, meeting item, or post was received. Read/write for the MeetingItem object; read-only for the MailItem and PostItem objects.

expression.ReceivedTime

expression Required. An expression that returns a MailItem, MeetingItem, or PostItem object.
RecipientReassignmentProhibited Property

True if the recipient cannot forward the mail message. Read/write Boolean.

expression.RecipientReassignmentProhibited

expression Required. An expression that returns a MailItem object.
Recipients Property

- Returns a **Recipients** collection that represents all the recipients for the Outlook item. Read-only.

*expression*.**Recipients**

*expression* Required. An expression that returns an **AppointmentItem**, **JournalItem**, **MailItem**, **MeetingItem**, or **TaskItem** object.
**Example**

This Visual Basic for Applications example creates a new mail message, uses the `Add` method to add "Allison Klein" as a To recipient, and displays the message.

```vba
Set myOlApp = CreateObject("Outlook.Application")
Set myItem = myOlApp.CreateItem(olMailItem)
Set myRecipient = myItem.Recipients.Add("Allison Klein")
myItem.Display
```

If you use VBScript, you do not create the `Application` object, and you cannot use named constants. This example shows how to perform the same task using VBScript.

```vbs
Set myItem = Application.CreateItem(0)
Set myRecipient = myItem.Recipients.Add("Allison Klein")
myItem.Display
```
RecurrenceState Property

- Returns an OlRecurrenceState constant indicating the recurrence property of the specified object. Read-only.

OlRecurrenceState can be one of these OlRecurrenceState constants.
olApptException
olApptMaster
olApptNotRecurring
olApptOccurrence

expression.RecurrenceState

eexpression Required. An expression that returns an AppointmentItem object.
**RecurrenceType Property**

Returns or sets an **OlRecurrenceType** constant specifying the frequency of occurrences for the recurrence pattern. Read/write.

OlRecurrenceType can be one of these OlRecurrenceType constants.  
* olRecursDaily  
* olRecursMonthly  
* olRecursMonthNth  
* olRecursWeekly  
* olRecursYearly  
* olRecursYearNth

```
expression.RecurrenceType
```

**expression** Required. An expression that returns a **RecurrencePattern** object.
Example

This Visual Basic for Applications example creates a task called "Oil Change" that recurs every three months and uses the `Regenerate` property to set it to regenerate after each recurrence.

```vbscript
Set myOlApp = CreateObject("Outlook.Application")
Set myItem = myOlApp.CreateItem(olTaskItem)
Set myPattern = myItem.GetRecurrencePattern
myPattern.RecurrenceType = olRecursMonthly
myPattern.Regenerate = True
myPattern.Interval = 3
myItem.Subject = "Oil Change"
myItem.Save
myItem.Display
```

If you use VBScript, you do not create the `Application` object, and you cannot use named constants. This example shows how to perform the same task using VBScript.

```vbscript
Set myItem = Application.CreateItem(3)
Set myPattern = myItem.GetRecurrencePattern
myPattern.RecurrenceType = 2
myPattern.Regenerate = True
myPattern.Interval = 3
myItem.Subject = "Oil Change"
myItem.Save
myItem.Display
```
ReferredBy Property

Returns or sets a **String** specifying the referral name entry for the contact. Read/write.

`expression.ReferredBy`

*expression* Required. An expression that returns a **ContactItem** object.
Regenerate Property

True if the task should be regenerated following this pass through the recurrence pattern. This property is used to control the regeneration of the task as each occurrence of a recurring task is completed. Read/write Boolean.

expression.Regenerate

expression Required. An expression that returns a RecurrencePattern object.
**Example**

This Visual Basic for Applications example creates a task called "Oil Change" that recurs every three months and uses the `Regenerate` property to set it to regenerate after each recurrence.

```vba
Set myOlApp = CreateObject("Outlook.Application")
Set myItem = myOlApp.CreateItem(olTaskItem)
Set myPattern = myItem.GetRecurrencePattern
myPattern.RecurrenceType = olRecursMonthly
myPattern.Regenerate = True
myPattern.Interval = 3
myItem.Subject = "Oil Change"
myItem.Save
myItem.Display
```

If you use VBScript, you do not create the `Application` object, and you cannot use named constants. This example shows how to perform the same task using VBScript.

```vba
Set myItem = Application.CreateItem(3)
Set myPattern = myItem.GetRecurrencePattern
myPattern.RecurrenceType = 2
myPattern.Regenerate = True
myPattern.Interval = 3
myItem.Subject = "Oil Change"
myItem.Save
myItem.Display
```
**ReminderMinutesBeforeStart Property**

Returns or sets a **Long** indicating the number of minutes the reminder should occur prior to the start of the appointment. Read/write.

*expression*.**ReminderMinutesBeforeStart**

*expression*  Required. An expression that returns an **AppointmentItem** object.
ReminderOverrideDefault Property

**True** if the reminder overrides the default reminder behavior for the appointment, mail item, or task. Read/write **Boolean**.

**Note** You must set the **ReminderOverrideDefault** property to validate the **ReminderPlaySound** and the **ReminderSoundFile** properties.

*expression*.**ReminderOverrideDefault**

*expression*  Required. An expression that returns an **AppointmentItem**, **MailItem**, or **TaskItem** object.
ReminderPlaySound Property

True if the reminder should play a sound when it occurs for this appointment or task. The ReminderPlaySound property must be set in order to validate the ReminderSoundFile property. Read/write Boolean.

Note This property is only valid if the ReminderOverrideDefault property is set to True.

equation ReminderPlaySound

expression Required. An expression that returns an AppointmentItem, MailItem, or TaskItem object.
Reminders Property

Returns a Reminders collection that represents all current reminders. Read-only.

expression.Reminders

type

expression Required. An expression that returns an Application object.
Example

The following example returns the Reminders collection and displays the captions of all reminders in the collection. If there are no current reminders, a message is displayed to the user.

Sub ViewReminderInfo()
'Lists reminder caption information
    Dim olApp As Outlook.Application
    Dim objRem As Reminder
    Dim objRems As Reminders
    Dim strTitle As String
    Dim strReport As String
    Set olApp = Outlook.Application
    Set objRems = olApp.Reminders
    strTitle = "Current Reminders:",
    'If there are reminders, display message
    If olApp.Reminders.Count <> 0 Then
        For Each objRem In objRems
            'If string is empty, create new string
            If strReport = "" Then
                strReport = objRem.Caption & vbCr
            Else
                'Add info to string
            End If
        Next objRem
        'Display report in dialog
        MsgBox strTitle & vbCrLf & vbCrLf & strReport
    Else
        MsgBox "There are no reminders in the collection."
    End If
End Sub
ReminderSet Property

True if a reminder has been set for this appointment, mail item or task. Read/write Boolean.

expression.ReminderSet

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

This example uses the **PropertyChange** event to disable the setting of a reminder on an item.

```vba
Sub Item_PropertyChange(ByVal myPropertyName)    
    Select Case myPropertyName
        Case "ReminderSet"
            MsgBox "You may not set a reminder on this item!"
            Item.ReminderSet = False
        Case Else
    End Select
End Sub
```
ReminderSoundFile Property

Returns or sets a String indicating the path and file name of the sound file to play when the reminder occurs for the appointment, mail message, or task. This property is only valid if the ReminderOverrideDefault and ReminderPlaySound properties are set to True. Read/write.

expression.ReminderSoundFile

expression  Required. An expression that returns an AppointmentItem, MailItem, or TaskItem object.
ReminderTime Property

Returns or sets a **Date** indicating the date and time at which the reminder should occur for the specified **item**. Read/write.

*expression.ReminderTime*

*expression* Required. An expression that returns a **MailItem**, **MeetingItem** or **TaskItem** object.
RemoteMessageClass Property

Returns a **String** indicating the message class for the remote item. Read-only.

`expression.RemoteMessageClass`

`expression`  Required. An expression that returns a **RemoteItem** object.
RemoteStatus Property

Returns or sets an OlRemoteStatus constant specifying the remote status of the mail message. Read/write.

OlRemoteStatus can be one of these OlRemoteStatus constants.
olMarkedForCopy
olMarkedForDelete
olMarkedForDownload
olRemoteStatusNone
olUnMarked

expression.RemoteStatus

expression Required. An expression that returns a MailItem object.
ReplyRecipientNames Property

Returns a semicolon-delimited String list of reply recipients for the mail message. This property only contains the display names for the reply recipients. The reply recipients list should be set by using the ReplyRecipients collection. Read-only.

expression.ReplyRecipientNames

expression Required. An expression that returns a MailItem object.
ReplyRecipients Property

Returns a Recipients collection that represents all the reply recipient objects for the mail message.

expression.ReplyRecipients

expression Required. An expression that returns a MailItem object.
ReplyStyle Property

- Returns or sets an `OlActionReplyStyle` constant indicating the text formatting reply style for the specified action. Read/write.

OlActionReplyStyle can be one of these OlActionReplyStyle constants:
- `olEmbedOriginalItem`
- `OlIncludeOriginalText`
- `olIndentOriginalText`
- `olLinkOriginalItem`
- `olOmitOriginalText`
- `olReplyTickOriginalText`
- `olUserPreference`

```
expression.ReplyStyle
```

- `expression` Required. An expression that returns an `Action` object.
ReplyTime Property

Returns or sets a Date indicating the reply time for the appointment. Read/write.

expression.ReplyTime

expression  Required. An expression that returns an AppointmentItem object.
RequiredAttendees Property

Returns a semicolon-delimited String of required attendee names for the meeting appointment. This property only contains the display names for the required attendees. The attendee list should be set by using the Recipients collection. Read/write.

expression.RequiredAttendees

expression Required. An expression that returns an AppointmentItem object.
Resolved Property

True if the recipient has been validated against the Address Book. Read-only Boolean.

expression.Resolved

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

This Visual Basic for Applications example uses the `GetSharedDefaultFolder` method to resolve the `Recipient` object representing Kim Buhler, and then returns her shared default Calendar folder.

```vba
Set myOlApp = CreateObject("Outlook.Application")
Set myNameSpace = myOlApp.GetNameSpace("MAPI")
Set myRecipient = myNameSpace.CreateRecipient("Kim Buhler")
myRecipient.Resolve
If myRecipient.Resolved Then
    Set KimCalendarFolder = _
        myNameSpace.GetSharedDefaultFolder _
        (myRecipient, olFolderCalendar)
End If
```

If you use VBScript, you do not create the `Application` object, and you cannot use named constants. This example shows how to perform the same task using VBScript.

```vbs
Set myNameSpace = Application.GetNameSpace("MAPI")
Set myRecipient = myNameSpace.CreateRecipient("Kim Buhler")
myRecipient.Resolve
If myRecipient.Resolved Then
    Set KimCalendarFolder = _
        myNameSpace.GetSharedDefaultFolder _
        (myRecipient, 9)
End If
```
Resources Property

- Returns a semicolon-delimited **String** of resource names for the meeting. This property contains the display names only. The **Recipients** collection should be used to modify the resource recipients. Resources are added as **BCC** recipients to the collection. Read/write.

*expression*.Resources

*expression*  Required. An expression that returns an **AppointmentItem** object.
ResponseRequested Property

**True** if the sender would like a response to the meeting request for the appointment. Read/write **Boolean**.

*expression*.ResponseRequested

*expression* Required. An expression that returns an **AppointmentItem** object.
ResponseState Property

- Returns or sets an **OlTaskResponse** constant indicating the overall status of the response to the specified task request. Read-only.

OlTaskResponse can be one of these OlTaskResponse constants.

- **olTaskAccept**
- **olTaskAssign**
- **olTaskDecline**
- **olTaskSimple**

**expression**.ResponseState

**expression**  Required. An expression that returns a **TaskItem** object.
ResponseStatus Property

- Returns an **OlResponseStatus** constant indicating the overall status of the meeting for the current user for the appointment. Read-only.

OlResponseStatus can be one of these OlResponseStatus constants.

- **olResponseAccepted**
- **olResponseDeclined**
- **olResponseNone**
- **olResponseNotResponded**
- **olResponseOrganized**
- **olResponseTentative**

$expression.ResponseStatus$

$expression$ Required. An expression that returns an **AppointmentItem** object.
**ResponseStyle Property**

Returns or sets an **OlActionResponseStyle** constant indicating the response style used when the specified action is executed. Read/write.

OlActionResponseStyle can be one of these OlActionResponseStyle constants.

- **olOpen**
- **olPrompt**
- **olSend**

`expression.ResponseStyle`

*expression*  Required. An expression that returns an **Action** object.
Results Property

Returns a Results collection that specifies the results of the search.

expression.Result

expression  Required. An expression that returns a Search object.
**Example**

The following event procedure example stores the results of a search in a variable called `objRsts` and prints the results of the search in the Immediate window. Always use the `AdvancedSearchComplete` event to ensure the results of the search.

```vba
Private Sub Application_AdvancedSearchComplete(ByVal SearchObject As Search)
    Dim objRsts As Results
    MsgBox "The search " & SearchObject.Tag & " has completed. The scope was " & SearchObject.Scope & "."
    Set objRsts = SearchObject.Results
    'Print out number in Results collection
    Debug.Print objRsts.Count
    'Print out each member of Results collection
    For Each Item In objRsts
        Debug.Print Item
    Next
End Sub
```
Role Property

- Returns or sets a **String** containing the free-form text string associating the owner of a task with a role for the task. Read/write.

  expression.Role

  expression  Required. An expression that returns a [TaskItem](#) object.
Saved Property

**True** if the Microsoft Outlook item has not been modified since the last save. Read-only **Boolean**.

*expression*.Saved

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

This example tests for the Close event and if the item has not been saved, it uses the Save method to save the item without prompting the user.

Function Item_Close()
    If not Item.Saved Then
        Item.Save
    End If
End Function
SaveOption Property

Returns an **OlViewSaveOption** constant that specifies the folders in which the specified view is available and the read permissions attached to the view. The **SaveOption** property is set when the **View** object is created by using the **Add** method. Read-only.

**OlViewSaveOption** can be one of these **OlViewSaveOption** constants.

- **olViewSaveOptionAllFoldersOfType** All folders of this type can use the view.
- **olViewSaveOptionThisFolderEveryone** All users who can access the current folder can use the view. The view is only associated with this folder.
- **olViewSaveOptionThisFolderOnlyMe** The view is only associated with the current folder and only the user can access the view.

**expression.SaveOption**

**expression**  Required. An expression that returns a **View** object.
Example

The following example displays the names of all views that can be accessed by all users in the Notes folder.

Sub DisplayPublicViews()
'Displays the names of all views that are available to 'all users of this folder

    Dim olApp As Outlook.Application
    Dim objName As NameSpace
    Dim objViews As Views
    Dim objView As View
    Dim strSaveOption As String

    Set olApp = Outlook.Application
    Set objName = olApp.GetNamespace("MAPI")
    Set objViews = objName.GetDefaultFolder(olFolderNotes).Views

    For Each objView In objViews
        If objView.SaveOption = olViewSaveOptionThisFolderEveryone Then
            If strSaveOption = "" Then
                strSaveOption = "The following views are available" " to all users:" & vbCr & vbCr & objView.Name & vbCr
            Else
                strSaveOption = strSaveOption & _
                objView.Name & vbCr
            End If
        End If
    Next objView

    If strSaveOption <> "" Then
        MsgBox strSaveOption
    End If

End Sub
SaveSentMessageFolder Property

Returns a MAPIFolder object that represents the folder in which a copy of the mail message will be saved upon being sent.

expression.SaveSentMessageFolder

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

This VBScript example uses the `Reply` event and sets the Sent Items folder for the reply item to the folder in which the original item resides.

```vbscript
Function Item_Reply(ByVal myResponse)
  Set myResponse.SaveSentMessageFolder = Item.Parent
End Function
```
SchedulePlusPriority Property

Returns or sets a **String** representing the Microsoft Schedule+ priority for the task. Can be 1 through 9, A through Z, or A1 through Z9. Priority 1 is the highest. Read/write.

**expression.SchedulePlusPriority**

**expression**  Required. An expression that returns a **TaskItem** object.
Scope Property

Returns a String that specifies the scope of the specified search. Read-only.

eexpression.Scope

eexpression   Required. An expression that returns a Search object.
Remarks

The scope of the search is defined when the search is initiated. For more information, see the AdvancedSearch method.
Example

The following subroutine creates a **Search** object. The user's Inbox is specified as the scope of the search. The event subroutine occurs when the search has completed and displays the **Tag** and **Scope** properties for the new object as well as the results of the search.

Sub SearchInboxFolder()
'
Searches the Inbox

    Dim objSch As Search
    Const strF As String = 
        "urn:schemas:mailheader:subject = 'Office Christmas Party'"
    Const strS As String = "Inbox"
    Const strTag As String = "SubjectSearch"
    Set objSch = Application.AdvancedSearch(Scope:=strS, _
        Filter:=strF, SearchSubFolders:=True, Tag:=strTag)

End Sub

The **AdvancedSearchComplete** event is used to capture the results of the search.

Private Sub Application_AdvancedSearchComplete(ByVal SearchObject As

    Dim objRsts As Results
    MsgBox "The search " & SearchObject.Tag & 
        "has completed. The scope of the search was " & 
        SearchObject.Scope & "."
    Set objRsts = SearchObject.Results
    'Print out number in results collection
    Debug.Print objRsts.Count
    'Print out each member of results collection
    For Each Item In objRsts
        Debug.Print Item
    Next

End Sub
ScriptText Property

Returns a String containing all the VBScript in the form's Script Editor. Read-only.

expression.ScriptText

description Required. An expression that returns a FormDescription object.
Example

This VBScript example uses the **Open** event to access the **HTMLBody** property of an item. This sets the **EditorType** property of the item’s **Inspector** to **olEditorHTML**. When the item's **Body** property is set, the **EditorType** property is changed to the default. For example, if the default editor is set to RTF, the **EditorType** is set to **olEditorRTF**.

If this code is placed in the Script Editor of a form in design mode, the message boxes during run time will reflect the change in the **EditorType** as the body of the form changes. The final message box utilizes the **ScriptText** property to display all the VBScript code in the Script Editor.

Function Item_Open()

' Set the HTMLBody of the item.
Item.HTMLBody = "<HTML><H2>My HTML page.</H2><BODY>My body.</BODY></HTML>"

' Item displays HTML message.
Item.Display

' MsgBox shows EditorType is 2.
MsgBox "HTMLBody EditorType is " & Item.GetInspector.EditorType

' Access the Body and show
'the text of the Body.
MsgBox "This is the Body: " & Item.Body

' After accessing, EditorType
' is still 2.
MsgBox "After accessing, the EditorType is " & Item.GetInspector.EditorType

' Set the item's Body property.
Item.Body = "Back to default body."

' After setting, EditorType is
' now back to the default.
MsgBox "After setting, the EditorType is " & Item.GetInspector.EditorType

' Access the items's
' FormDescription object.
Set myForm = Item.FormDescription

' Display all the code
'in the Script Editor.
MsgBox myForm.ScriptText

End Function
SearchSubFolders Property

Returns a **Boolean** indicating whether the scope of the specified search included the subfolders of any folders searched. This property is determined by the `SearchSubfolders` argument of the `AdvancedSearch` method and is specified when the search is initiated. Read-only.

`expression.SearchSubFolders`

`expression`  Required. An expression that returns a **Search** object.
Remarks

If True, the Search object searched through any subfolders in the specified filter path.
**Example**

The following example creates a `Search` object that contains all items in the user's Inbox with the subject Office Christmas Party. The `SearchSubFolders` argument is specified as `True`.

```vba
Sub SearchInboxFolder()
    'Searches the Inbox
    Dim objSch As Search
    Const strF As String = _
        "urn:schemas:mailheader:subject = 'Office Christmas Party'",
    Const strS As String = "Inbox"
    Const strTag As String = "SubjectSearch"
    Set objSch = Application.AdvancedSearch(Scope:=strS, _
        Filter:=strF, SearchSubFolders:=True, Tag:=strTag)
End Sub

Use an **AdvancedSearchComplete** event procedure to display the results of the search.

```vba
Private Sub Application_AdvancedSearchComplete(ByVal SearchObject As Search)
    Dim objRsts As Results
    MsgBox "The search " & SearchObject.Tag & " has completed. The SearchSubFolders was set to " & SearchObject.SearchSubFolders & "."
    Set objRsts = SearchObject.Results
    'Print out number in Results collection
    Debug.Print objRsts.Count
    'Print out each member of Results collection
    For Each Item In objRsts
        Debug.Print Item
    Next
End Sub
```
SelectedMailingAddress Property

Returns or sets an OlMailingAddress constant indicating the type of the mailing address for the contact. Read/write.

OlMailingAddress can be one of these OlMailingAddress constants:

- **olBusiness**
- **olHome**
- **olNone**
- **olOther**

`expression.SelectedMailingAddress`

`expression`  Required. An expression that returns a ContactItem object.
Selection Property

- Returns a Selection object consisting of one or more items selected in the current view.

expression.Selection

expression Required. An expression that returns an Explorer object.
Remarks

If the current folder is a file-system folder, or if Outlook Today or any folder with a current Web view is currently displayed, this property returns an empty collection.
Example

The following Microsoft Visual Basic/Visual Basic for Applications example uses the **Count** property and **Item** method of the **Selection** collection returned by the **Selection** property to display the senders of all messages selected in the active explorer window.

```vba
Dim myOlApp As New Outlook.Application
Dim myOlExp As Outlook.Explorer
Dim myOlSel As Outlook.Selection
Dim MsgTxt As String
MsgTxt = "You have selected items from" & Chr(13) & Chr(13)
Set myOlExp = myOlApp.ActiveExplorer
Set myOlSel = myOlExp.Selection
For x = 1 To myOlSel.Count
    MsgTxt = MsgTxt & myOlSel.Item(x).SenderName & Chr(13)
Next x
MsgBox MsgTxt
```

If you use VBScript, you do not create the **Application** object. This example shows how to perform the same task using VBScript.

```vbs
MsgTxt = "You have selected items from" & Chr(13) & Chr(13)
Set myOlSel = Application.ActiveExplorer.Selection
For x = 1 To myOlSel.Count
    MsgTxt = MsgTxt & myOlSel.Item(x).SenderName & Chr(13)
Next
MsgBox MsgTxt
```
SenderName Property

Returns a String indicating the display name of the sender for the mail message, meeting item or post. This property corresponds to the MAPI property PR_SENDER_NAME. Read-only.

expression.SenderName

expression Required. An expression that returns a MailItem, MeetingItem or PostItem object.
Sensitivity Property

Returns or sets an **OISensitivity** constant indicating the sensitivity for the Microsoft Outlook item. This property corresponds to the MAPI property PR_SENSITIVITY. Read/write.

OISensitivity can be one of these OISensitivity constants.

- **olConfidential**
- **olNormal**
- **olPersonal**
- **olPrivate**

**expression.Sensitivity**

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

Returns a **Boolean** value that indicates if a message has been sent. **True** if sent, **False** if not sent. Read-only.

In general, there are three different kinds of messages: sent, posted, and saved. Sent messages are traditional e-mail messages or meeting items sent to a recipient or public folder. Posted messages are created in a public folder. Saved messages are created and saved without either sending or posting.

**expression.Sent**

*expression*  Required. An expression that returns a **MeetingItem** or **MailItem** object.
SentOn Property

Returns a Date indicating the date and time on which the mail message, meeting item, or post was sent. This property corresponds to the MAPI property PR_CLIENT_SUBMIT_TIME. When you send a meeting request item using the object's Send method, the transport provider sets the ReceivedTime and SentOn properties for you. Read-only.

expression.SentOn

expression Required. An expression that returns a MailItem, MeetingItem or PostItem object.
SentOnBehalfOfName Property

Returns a String indicating the display name for the intended sender of the mail message. This property corresponds to the MAPI property PR_SENT_REPRESENTING_NAME. Read/write.

expression.SentOnBehalfOfName

expression  Required. An expression that returns a MailItem object.
Session Property

- Returns the NameSpace object for the current session session.

expression.Session

expression  Required. An expression that returns one of the objects in the Applies To list to which the Session property is being applied.
Shortcuts Property

- Returns an `OutlookBarShortcuts` collection of `shortcuts` contained within the specified `Outlook Bar` group.

`expression.Shortcuts`

`expression`  Required. An expression that returns an `OutlookBarGroup` object.
**Example**

This Microsoft Visual Basic/Visual Basic for Applications example deletes all empty groups in the Outlook Bar.

```vba
Dim myOlApp As New Outlook.Application
Dim myOlBar As Outlook.OutlookBarPane
Dim myOlGroup As Outlook.OutlookBarGroup
Set myOlBar = myOlApp.ActiveExplorer.Panes.Item("OutlookBar")
For x = myOlBar.Contents.Groups.Count To 1 Step -1
    Set myOlGroup = myOlBar.Contents.Groups.Item(x)
    If myOlGroup.Shortcuts.Count = 0 Then
        myOlBar.Contents.Groups.Remove x
    End If
Next x
```

If you use VBScript, you do not create the `Application` object. This example shows how to perform the same task using VBScript.

```vbs
Set myOlBar = Application.ActiveExplorer.Panes.Item("OutlookBar")
For x = myOlBar.Contents.Groups.Count To 1 Step -1
    Set myOlGroup = myOlBar.Contents.Groups.Item(x)
    If myOlGroup.Shortcuts.Count = 0 Then
        myOlBar.Contents.Groups.Remove x
    End If
Next x
```
ShowAsOutlookAB Property

Returns or sets a read-only **Boolean** variable that specifies whether the folder will be displayed as an Outlook Address Book. Read-write.

*expression*.ShowAsOutlookAB

*expression*  Required. An expression that returns a **MAPIFolder** object.
Example

The following example creates a reference to a new folder and modifies its ShowAsOutlookAB property so that it is displayed as an Address Book.

Sub FaveChange()

    Dim olApp As Outlook.Application
    Dim nmsName As namespace
    Dim fldFolder As MAPIFolder

    Set olApp = Outlook.Application
    'Create instance of namespace
    Set nmsName = olApp.GetNamespace("Mapi")
    Set fldFolder = nmsName.GetDefaultFolder(olFolderInbox)

    'Display the folder as Outlook Address Book
    fldFolder.ShowAsOutlookAB = True

End Sub
ShowOn Property

- Returns or sets a **OlActionShowOn** constant representing the location where the action will be shown. Read/write.

**OlActionShowOn** can be one of these **OlActionShowOn** constants.

- **olDontShow**
- **olMenu**
- **olMenuAndToolbar**

`expression.ShowOn`

`expression` Required. An expression that returns an **Action** object.
Size Property

Returns a Long indicating the size (in bytes) of the Outlook item. Read-only.

expression.Size

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

Returns or sets a **String** indicating the spouse name entry for the contact. Read/write.

*expression*.Spouse

*expression*  Required. An expression that returns a **ContactItem** object.
Standard Property

- Returns a **Boolean** that specifies whether or not the view is a built-in Microsoft Outlook view.

*expression*.Standard

*expression* Required. An expression that returns a **View** object.
Start Property

- Returns or sets a Date indicating the starting date and time for the appointment or journal entry. Read/write.

expression.Start

expression Required. An expression that returns an AppointmentItem or JournalItem object.
Example

This Visual Basic for Applications example uses **CreateItem** to create an appointment and uses **MeetingStatus** to set the meeting status to "Meeting" to turn it into a meeting request with both a required and an optional attendee.

```vba
Set myOlApp = CreateObject("Outlook.Application")
Set myItem = myOlApp.CreateItem(olAppointmentItem)
myItem.MeetingStatus = olMeeting
myItem.Subject = "Strategy Meeting"
myItem.Location = "Conference Room B"
myItem.Start = #9/24/97 1:30:00 PM#
myItem.Duration = 90
Set myRequiredAttendee = myItem.Recipients.Add ("Nate Sun")
myRequiredAttendee.Type = olRequired
Set myOptionalAttendee = myItem.Recipients.Add ("Kevin Kennedy")
myOptionalAttendee.Type = olOptional
Set myResourceAttendee = myItem.Recipients.Add("Conference Room B")
myResourceAttendee.Type = olResource
myItem.Display

If you use VBScript, you do not create the **Application** object, and you cannot use named constants. This example shows how to perform the same task using VBScript.

```vbs
Set myItem = Application.CreateItem(1)
myItem.MeetingStatus = 1
myItem.Subject = "Strategy Meeting"
myItem.Location = "Conference Room B"
myItem.Start = #9/24/97 1:30:00 PM#
myItem.Duration = 90
Set myRequiredAttendee = myItem.Recipients.Add ("Nate Sun")
myRequiredAttendee.Type = 1
Set myOptionalAttendee = myItem.Recipients.Add ("Kevin Kennedy")
myOptionalAttendee.Type = 2
Set myResourceAttendee = myItem.Recipients.Add("Conference Room B")
myResourceAttendee.Type = 3
myItem.Display
```
StartDate Property

Returns or sets a Date specifying the starting date and time for the specified task. Read/write.

expression.StartDate

description Required. An expression that returns a TaskItem object.
StartTime Property

Returns or sets a Date indicating the start time for a given occurrence of the recurrence pattern. This property is only valid for appointments. Read/write.

expression.StartTime

dateexpression Required. An expression that returns a RecurrencePattern object.
Status Property

Returns or sets an **OlTaskStatus** constant specifying the status for the task. Corresponds to the **Status** field of a **TaskItem**. Read/write.

OlTaskStatus can be one of these OlTaskStatus constants.

- **olTaskComplete**
- **olTaskDeferred**
- **olTaskInProgress**
- **olTaskNotStarted**
- **olTaskWaiting**

```
expression.Status
```

*expression*  Required. An expression that returns a **TaskItem** object.
StatusOnCompletionRecipients

Property

Returns or sets a semicolon-delimited String of display names for recipients who will receive status upon completion of the task. This property is calculated from the Recipients property. Recipients returned by the StatusOnCompletionRecipients property correspond to BCC recipients in the Recipients collection. Read/write.

expression.StatusOnCompletionRecipients

expression Required. An expression that returns a TaskItem object.
StatusUpdateRecipients Property

Returns a semicolon-delimited String of display names for recipients who receive status updates for the task. This property is calculated from the Recipients property. Recipients returned by the StatusUpdateRecipients property correspond to CC recipients in the Recipients collection. Read-only.

expression.StatusUpdateRecipients

expression  Required. An expression that returns a TaskItem object.
StoreID Property

Returns a String indicating the store ID for the folder. Read-only.

`expression.StoreID`

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

This Visual Basic for Applications example obtains the **EntryID** and **StoreID** for a subfolder within the Tasks **Folders** collection and then calls the **GetFolderFromID** method using these values to obtain the same subfolder. The folder is then displayed.

```vba
Set myOlApp = CreateObject("Outlook.Application")
Set myNameSpace = myOlApp.GetNameSpace("MAPI")
Set myFolder = myNameSpace.GetDefaultFolder(olFolderTasks)
Set myOwnFolder = myFolder.Folders(1)
myEntryID = myOwnFolder.EntryID
myStoreID = myOwnFolder.StoreID
Set myNewFolder = myNameSpace.GetFolderFromID(myEntryID, myStoreID)
myNewFolder.Display
```

If you use VBScript, you do not create the **Application** object, and you cannot use named constants. This example shows how to perform the same task using VBScript.

```vba
Set myNameSpace = Application.GetNameSpace("MAPI")
Set myFolder = myNameSpace.GetDefaultFolder(13)
Set myOwnFolder = myFolder.Folders(1)
myEntryID = myOwnFolder.EntryID
myStoreID = myOwnFolder.StoreID
Set myNewFolder = myNameSpace.GetFolderFromID(myEntryID, myStoreID)
myNewFolder.Display
```
Subject Property

Returns or sets a String indicating the subject for the Microsoft Outlook item. This property corresponds to the MAPI property PR_SUBJECT. The Subject property is the default property for Outlook items. Read/write.

**Note** For a NoteItem object, the Subject property is a read-only String that is calculated from the body text of the note.

`expression.Subject`

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

This Visual Basic for Applications example creates a mail message, sets the **Subject** to "Speeches", uses the **Copy** method to copy it, then moves the copy into a newly-created mail folder named "Saved Mail" within the Tasks folder.

```vba
Set myOlApp = CreateObject("Outlook.Application")
Set myNamespace = myOlApp.GetNamespace("MAPI")
Set myFolder = myNamespace.GetDefaultFolder(olFolderInbox)
Set myNewFolder = myFolder.Folders.Add("Saved Mail", olFolderDrafts)  
Set myItem = myOlApp.CreateItem(olMailItem)
    myItem.Subject = "Speeches"
Set myCopiedItem = myItem.Copy
myCopiedItem.Move myNewFolder
```

If you use VBScript, you do not create the `Application` object, and you cannot use named constants. This example shows how to perform the same task using VBScript.

```vbs
Set myNamespace = Application.GetNamespace("MAPI")
Set myFolder = myNamespace.GetDefaultFolder(6)
Set myNewFolder = myFolder.Folders.Add("Saved Mail", 16)
Set myItem = Application.CreateItem(0)
    myItem.Subject = "Speeches"
Set myCopiedItem = myItem.Copy
myCopiedItem.Move myNewFolder
```
Show All
Submitted Property

Returns a **Boolean** value that indicates if the item has been submitted. **True** if the item has been submitted. A message is always created and submitted in a folder, usually the Outbox. Read-only.

*expression*.Submitted

*expression*  Required. An expression that returns a **MeetingItem** or **MailItem** object.
Suffix Property

Returns or sets a **String** indicating the name suffix (such as Jr., III, or Ph.D.) for the specified contact. Read/write.

`expression.Suffix`

`expression` Required. An expression that returns a **ContactItem** object.

**Note** The **LastName**, **FirstName**, **MiddleName**, and **Suffix** properties are parsed from the **FullName** property.
SyncObjects Property

- Returns a SyncObjects collection containing all synchronization profiles.

dot SyncObjects

dot Required. An expression that returns a NameSpace object.
Example

This Microsoft Visual Basic/Visual Basic for Applications example starts synchronization of the user’s second synchronization profile in response to the user clicking a button on a form.

Private Sub Command1_Click()
    Dim myOlApp As New Outlook.Application
    Dim mySyncs As Outlook.SyncObjects
    Set mySyncs = myOlApp.GetNamespace("MAPI").SyncObjects
    Set mySyncObj = mySyncs.Item(1)
    mySyncObj.Start
End Sub
Tag Property

Returns a **String** specifying the name of the current search. The **Tag** property is used to identify a specific search.

`expression.Tag`

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

The **Tag** property is set by the *AdvancedSearch* method when the *Search* object is created.
**Example**

The following example searches through the user's inbox for all items that do not have a flag. The name "FlagSearch", specified by the Tag property, is given to the search.

Sub SearchForFlags()
    'List all items in the Inbox that do NOT have a flag:
    Dim objSch As Search
    Const strF As String = "urn:schemas:httpmail:messageflag = 0" & 
        " OR urn:schemas:httpmail:messageflag IS NULL"
    Const strS As String = "Inbox"
    Set objSch = _
        Application.AdvancedSearch(Scope:=strS, Filter:=strF, Tag:="FlagSearch"
End Sub

The following example uses the **AdvancedSearchComplete** event to capture the outcome of the search. The Tag property of the search is displayed to notify the user which search has completed.

Private Sub Application_AdvancedSearchComplete(ByVal SearchObject As Search)
    Dim objRsts As Results
    MsgBox "The search " & SearchObject.Tag & " has completed."
    Set objRsts = SearchObject.Results
    'Print out number in Results collection
    Debug.Print objRsts.Count
    'Print out each member of Results collection
    For Each Item In objRsts
        Debug.Print Item
    Next
End Sub
Target Property

...-

Returns a **Variant** indicating the target of the specified shortcut in an **Outlook Bar** group. Read-only.

*expression*.Target

*expression*  Required. An expression that returns an **OutlookBarShortcut** object.
Remarks

The return type depends on the shortcut type. If the shortcut represents a Microsoft Outlook folder, the return type is MAPIFolder. If the shortcut represents a file-system folder, the return type is an Object. If the shortcut represents a file-system path or URL, the return type is a String.
Example

This Microsoft Visual Basic/Visual Basic for Applications example steps through the shortcuts in the first Outlook Bar group. If it finds a shortcut that is not an Outlook folder, it deletes it.

Dim myOlApp As New Outlook.Application
Dim myOlBar As Outlook.OutlookBarPane
Dim myolGroup As Outlook.OutlookBarGroup
Dim myOlShortcuts As Outlook.OutlookBarShortcuts
Dim myOlShortcut As Outlook.OutlookBarShortcut
Set myOlBar = myOlApp.ActiveExplorer.Panes.Item("OutlookBar")
Set myolGroup = myOlBar.Contents.Groups.Item(1)
Set myOlShortcuts = myolGroup.Shortcuts
myTop = myOlShortcuts.Count
For x = myTop To 1 Step -1
  Set myOlShortcut = myOlShortcuts.Item(x)
  If TypeName(myOlShortcut.Target) <> "MAPIFolder" Then
    myOlShortcuts.Remove x
  End If
Next x

If you use VBScript, you do not create the Application object. This example shows how to perform the same task using VBScript.

Set myOlBar = _
            Application.ActiveExplorer.Panes.Item("OutlookBar")
Set myolGroup = myOlBar.Contents.Groups.Item(1)
Set myOlShortcuts = myolGroup.Shortcuts
myTop = myOlShortcuts.Count
For x = myTop To 1 Step -1
  Set myOlShortcut = myOlShortcuts.Item(x)
  If TypeName(myOlShortcut.Target) <> "MAPIFolder" Then
    myOlShortcuts.Remove x
  End If
Next x
**TeamTask Property**

*True* if the task is a team task. Read/write *Boolean*.

*expression.TeamTask*

*expression*  Required. An expression that returns a *TaskItem* object.
TelexNumber Property

Returns or sets a **String** indicating the telex number for the contact. Read/write.

`expression.TelexNumber`

*expression* Required. An expression that returns a **ContactItem** object.
Template Property

Returns or sets a String indicating the name of the template for the form. Read/write.

**Note** This property refers to a Microsoft Word .DOT template file, and so is applicable for forms with UseWordMail = True.

`expression.Template`

*expression* Required. An expression that returns a FormDescription object.
Title Property

Returns or sets a **String** indicating the title for the contact. Read/write.

`expression.Title`

`expression` Required. An expression that returns a **ContactItem** object.
To Property

Returns or sets a semicolon-delimited String list of display names for the To recipients for the Outlook item. This property contains the display names only. The **To** property corresponds to the MAPI property PR_DISPLAY_TO. The **Recipients** collection should be used to modify this property. Read/write.

`expression.To`  

`expression` Required. An expression that returns a **MailItem** object.
Top Property

Returns or sets a Long indicating the position (in pixels) of the top horizontal edge of an explorer, inspector, or note window from the edge of the screen. Read/write.

expression.Top

expression  Required. An expression that returns an Explorer, Inspector, or NoteItem object.
TotalWork Property

Returns or sets a Long indicating the total work for the task. Corresponds to the Total work field on the Details tab of a Task item. Read/write.

expression.TotalWork

expression  Required. An expression that returns a TaskItem object.
TrackingStatus Property

Returns or sets an **OlTrackingStatus** constant indicating the tracking status for the recipient. Read/write.

OlTrackingStatus can be one of these OlTrackingStatus constants.

- *olTrackingDelivered*
- *olTrackingNone*
- *olTrackingNotDelivered*
- *olTrackingNotRead*
- *olTrackingRead*
- *olTrackingRecallFailure*
- *olTrackingRecallSuccess*
- *olTrackingReplied*

*expression*.TrackingStatus

*expression* Required. An expression that returns a **Recipient** object.
TrackingStatusTime Property

Returns or sets a Date indicating the tracking status date and time for the recipient. Read/write.

expression.TrackingStatusTime

expression  Required. An expression that returns a Recipient object.
TransferSize Property

Returns a Long specifying the transfer size (in bytes) for the remote item. Read-only.

expression.TransferSize

class expression  Required. An expression that returns a RemoteItem object.
TransferTime Property

Returns a Long indicating the transfer time (in seconds) for the remote item. Read-only.

expression.TransferTime

expression Required. An expression that returns a RemoteItem object.
TTYTDDTelephoneNumber Property

- Returns or sets a String specifying the TTY/TDD telephone number for the contact. Read/write.

expression.TTYTDDTelephoneNumber

expression Required. An expression that returns a ContactItem object.
Type Property

- Type property as it applies to the Attachment object.

Returns an OlAttachmentType constant indicating the type of the specified object. Read-only.

OlAttachmentType can be one of these OlAttachmentType constants.

- olByReference
- olByValue
- olEmbeddedItem
- olOLE

expression.Type

expression Required. An expression that returns an Attachment object.

- Type property as it applies to the Link object.

Returns an OlObjectClass constant indicating the type of item represented by the Link object. Read-only.

OlObjectClass can be one of these OlObjectClass constants.

- olAction
- olActions
- olAddressEntries
- olAddressEntry
- olAddressList
- olAddressLists
- olApplication
- olAppointment
olAttachment
olAttachments
olContact
olDistributionList
olDocument
olException
olExceptions
olExplorer
olExplorers
olFolder
olFolders
olFormDescription
olInspector
olInspectors
olItemProperties
olItemProperty
olItems
olJournal
olLink
olLinks
olMail
olMeetingCancellation
olMeetingRequest
olMeetingResponseNegative
olMeetingResponsePositive
olMeetingResponseTentative
olNamespace
olNote
olObjects
olOutlookBarGroup
olOutlookBarGroups
olOutlookBarPane
olOutlookBarShortcut
expression.Type

expression  Required. An expression that returns a Link object.

- Type property as it applies to the ItemProperty and UserProperty objects.
Returns an OlUserPropertyType constant indicating the type of the specified object. Read-only.

OlUserPropertyType can be one of these OlUserPropertyType constants.

- olCombination
- olCurrency
- olDateTime
- olDuration
- olFormula
- olKeywords
- olNumber
- olOutlookInternal
- olPercent
- olText
- olYesNo

expression.Type

depends on the type of recipient, returns or sets a Long corresponding to the numeric equivalent of one of the following constants:

- **JournalItem** recipient: the OlJournalRecipientType constant olAssociatedContact.

- **MailItem** recipient: one of the following OlMailRecipientType constants: olBCC, olCC, olOriginator, or olTo.

- **MeetingItem** recipient: one of the following OlMeetingRecipientType constants: olOptional, olOrganizer, olRequired, or olResource.

- **TaskItem** recipient: either of the following OlTaskRecipientType constants: olFinalStatus, or olUpdate.
This property is read/write.

**expression.Type**

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

- **Type property as it applies to the **AddressEntry** and **JournalItem** objects.**

Returns or sets a **String** representing the type of entry for this address such as an Internet Address, MacMail Address, or Microsoft Mail Address (for the **AddressEntry** object), or a free-form **String** field, usually containing the display name of the journalizing application (for example, "MSWord") (for the **JournalItem** object). Read/write.

**expression.Type**

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

- **Type property as it applies to the **NameSpace** object.**

Returns a **String** indicating the type of the specified object. The only supported string is "MAPI." Read-only.

**expression.Type**

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

**Example**

This Visual Basic for Applications example uses **CreateItem** to create an appointment and uses **MeetingStatus** to set the meeting status to "Meeting" to turn it into a meeting request with both a required and an optional attendee.

```vba
Set myOlApp = CreateObject("Outlook.Application")
Set myItem = myOlApp.CreateItem(olAppointmentItem)
myItem.MeetingStatus = olMeeting
myItem.Subject = "Strategy Meeting"
myItem.Location = "Conference Room B"
myItem.Start = #9/24/97 1:30:00 PM#
myItem.Duration = 90
Set myRequiredAttendee = myItem.Recipients.Add ("Nate Sun")
myRequiredAttendee.Type = olRequired
```
Set myOptionalAttendee = myItem.Recipients.Add("Kevin Kennedy")
myOptionalAttendee.Type = olOptional
Set myResourceAttendee = myItem.Recipients.Add("Conference Room B")
myResourceAttendee.Type = olResource
myItem.Send

If you use VBScript, you do not create the Application object, and you cannot use named constants. This example shows how to perform the same task using VBScript.

Set myItem = Application.CreateItem(1)
myItem.MeetingStatus = 1
myItem.Subject = "Strategy Meeting"
myItem.Location = "Conference Room B"
myItem.Start = #9/24/97 1:30:00 PM#
myItem.Duration = 90
Set myRequiredAttendee = myItem.Recipients.Add("Nate Sun")
myRequiredAttendee.Type = 1
Set myOptionalAttendee = myItem.Recipients.Add("Kevin Kennedy")
myOptionalAttendee.Type = 2
Set myResourceAttendee = myItem.Recipients.Add("Conference Room B")
myResourceAttendee.Type = 3
myItem.Send
UnRead Property

True if the Microsoft Outlook item has not been opened (read). Read/write Boolean.

expression.UnRead

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

- Returns a **Long** indicating the number of unread items in the folder. Read-only.

```
expression.UnreadItemCount
```

*expression*  Required. An expression that returns a [MAPIFolder](#) object.
**User1 Property**

Returns or sets a **String** specifying the first Microsoft Schedule+ user for the contact. Read/write.

`expression.User1`

*expression*  Required. An expression that returns a [ContactItem](#) object.
User2 Property

Returns or sets a String specifying the second Microsoft Schedule+ user for the contact. Read/write.

expression.User2

description Required. An expression that returns a ContactItem object.
User3 Property

Returns or sets a **String** specifying the third Microsoft Schedule+ user for the contact. Read/write.

*expression*.User3

*expression*  Required. An expression that returns a **ContactItem** object.
**User4 Property**

- Returns or sets a **String** specifying the fourth Microsoft Schedule+ user for the contact. Read/write.

*expression*.User4

*expression*  Required. An expression that returns a **ContactItem** object.
UserCertificate Property

- Returns or sets a **String** containing the user's authentication certificate for the contact. Read/write.

```
expression.UserCertificate
```

- **expression**  Required. An expression that returns a **ContactItem** object.
UserProperties Property

Returns the UserProperties collection that represents all the user properties for the Outlook item.

`expression.UserProperties`  

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

This VBScript example uses the `CustomPropertyChange` event to enable a control when a **Boolean field** is set to **True**.

For this example, create two custom fields on the second page of a form. The first, a **Boolean** field is named "RespondBy". The second field is named "DateToRespond".

```vbscript
Sub Item_CustomPropertyChange(ByVal myPropName)
    Select Case myPropName
        Case "RespondBy"
            Set myPages = Item.GetInspector.ModifiedFormPages
            Set myCtrl = myPages("P.2").Controls("DateToRespond")
            If Item.UserProperties("RespondBy").Value Then
                myCtrl.Enabled = True
                myCtrl.Backcolor = 65535 'Yellow
            Else
                myCtrl.Enabled = False
                myCtrl.Backcolor = 0 'Black
            End If
        Case Else
    End Select
End Sub
```
UseWordMail Property

True to use Microsoft Word as the default editor for the form. Read/write Boolean.

expression.UseWordMail

expression Required. An expression that returns a FormDescription object.
ValidationFormula Property

Returns or sets a String indicating the validation formula for the user property. This property is just a descriptive string; it does not have any validation functionality. Read/write.

expression.ValidationFormula

expression Required. An expression that returns a UserProperty object.
ValidationText Property

- Returns or sets a String specifying the validation text for the specified user property. Read/write.

expression.ValidationText

expression Required. An expression that returns a UserProperty object.
Value Property

Returns or sets a **Variant** indicating the value for the specified user property. Read/write.

`expression.Value`

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

- Returns or sets a **String** indicating the number of the version. Read/write for the **FormDescription** object; read-only for all other objects in the Applies To list.

`expression.Version`

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

Returns a **Views** collection. Each **View** object in the **Views** collection defines the manner in which items are displayed in an Explorer.

*expression*.Views

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

The following example creates an instance of the Views collection and returns a view called "Table View".

Sub GetView()
'Returns a view object

    Dim olApp As Outlook.Application
    Dim objName As NameSpace
    Dim objViews As Views
    Dim objView As View

    Set olApp = Outlook.Application
    Set objName = olApp.GetNamespace("MAPI")
    Set objViews = objName.GetDefaultFolder(olFolderInbox).Views
    'Return a view called Table View
    Set objView = objViews.Item("Table View")

End Sub
**ViewType Property**

- **ViewType property as it applies to the OutlookBarGroup object.**

Returns or sets the icon view displayed by the specified Outlook bar group. Read/write **OlOutlookBarViewType**.

OlOutlookBarViewType can be one of these OlOutlookBarViewType constants.
- **olLargeIcon**
- **olSmallIcon**

*expression*.**ViewType*

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

- **ViewType property as it applies to the View object.**

Returns an **OlViewType** constant that represents the type of the current view. Read-only.

OlViewType can be one of these OlViewType constants.
- **olCalendarView**
- **olCardView**
- **olIconView**
- **olTableView**
- **olTimelineView**

*expression*.**ViewType*

*expression*  Required. An expression that returns a **View** object.
Example

As it applies to the OutlookBarGroup object.

The following Microsoft Visual Basic/Visual Basic for Applications example toggles the first Outlook bar group between large and small icon views.

Dim myOlApp As New Outlook.Application
Dim myOlBar As Outlook.OutlookBarPane
Dim myOlGroup As Outlook.OutlookBarGroup
Set myOlBar = myOlApp.ActiveExplorer.Panes.Item("OutlookBar")
Set myOlGroup = myOlBar.Contents.Groups.Item(1)
If myOlGroup.ViewType = olLargeIcon Then
    myOlGroup.ViewType = olSmallIcon
Else
    myOlGroup.ViewType = olLargeIcon
End If

If you use VBScript, you do not create the Application object, and you cannot use named constants. This example shows how to perform the same task using VBScript.

Set myOlBar = Application.ActiveExplorer.Panes.Item("OutlookBar")
Set myOlGroup = myOlBar.Contents.Groups.Item(1)
If myOlGroup.ViewType = 0 Then
    myOlGroup.ViewType = 1
Else
    myOlGroup.ViewType = 0
End If

As it applies to the View object.

The following example displays the name and type of all views in the user's inbox.

Sub DisplayViewMode()
'Displays the names and view modes for all views
    Dim olApp As Outlook.Application
    Dim objName As NameSpace
    Dim objViews As Views
    Dim objView As View
Dim strTypes As String

Set olApp = Outlook.Application
Set objName = olApp.GetNamespace("MAPI")
Set objViews = objName.GetDefaultFolder(olFolderInbox).Views

'Collect names and view types for all views
For Each objView In objViews
    strTypes = strTypes & objView.Name & vbTab & vbTab & objView.
Next objView

'Display message box
MsgBox "Current Inbox Views and Viewtypes:" & vbCrLf & _
    vbCrLf & strTypes

End Sub
Visible Property

Returns or sets a **Boolean** indicating the visible state of the specified object. **True** to display the object; **False** to hide the object. Read/write.

\[ \text{expression}.\text{Visible} \]

\[ \text{expression} \] Required. An expression that returns one of the items in the Applies To list.
Remarks

You can also use the `ShowPane` method or the `IsPaneVisible` method of an `Explorer` object to set or retrieve this value.
Example

This Microsoft Visual Basic/Visual Basic for Applications example toggles the visible state of the Outlook Bar.

Dim myOlApp As New Outlook.Application
Dim myOlBar As Outlook.OutlookBarPane
Set myOlBar = myOlApp.ActiveExplorer.Panes.Item("OutlookBar")
myOlBar.Visible = Not myOlBar.Visible

If you use VBScript, you do not create the Application object. This example shows how to perform the same task using VBScript.

Set myOlBar = Application.ActiveExplorer.Panes.Item("OutlookBar")
myOlBar.Visible = Not myOlBar.Visible
VotingOptions Property

Returns or sets a **String** specifying a delimited string containing the voting options for the mail message. Read/write.

`expression.VotingOptions`  

`expression` Required. An expression that returns a **MailItem** object.
VotingResponse Property

Returns or sets a String specifying the voting response for the mail message. This property is usually set to one of the delimited values returned by the VotingOptions property on a reply to the original message. Read/write.

*expression*.VotingResponse

*expression* Required. An expression that returns a MailItem object.
WebPage Property

Returns or sets a String indicating the URL of the Web page for the contact. Read/write.

expression.WebPage

expression  Required. An expression that returns a ContactItem object.
WebViewOn Property

Returns or sets a **Boolean** indicating the Web **view** state for a folder. **True** to display the Web page specified by the **WebViewURL** property. Read/write.

`expression.WebViewOn`

`expression` Required. An expression that returns a **MAPIFolder** object.
Remarks

This property is always \textbf{False} if the value of the \texttt{WebViewURL} property is empty.
WebViewURL Property

- Returns or sets a **String** indicating the **URL** of the Web page that is assigned to a folder. Read/write.

`expression.WebViewURL`

`expression`  Required. An expression that returns a **MAPIFolder** object.
Width Property

- Returns or sets a Long indicating the width (in pixels) of the specified object. Read/write.

_object_.Width

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

Returns or sets an **OlWindowState** constant specifying the window state of an explorer or inspector window. Read/write.

OlWindowState can be one of these OlWindowState constants.

- **olMaximized**
- **olMinimized**
- **olNormalWindow**

**object hWndState**

**object**  Required. An expression that returns an **Explorer** or **Inspector** object.
Example

This Microsoft Visual Basic/Visual Basic for Applications example minimizes all open explorer windows. It uses the **Count** property and **Item** method of the **Explorers** collection to enumerate the open explorer windows.

```vba
Dim myOlApp As New Outlook.Application
Dim myOlExp As Outlook.Explorer
Dim myOlExps As Outlook.Explorers
Set myOlExps = myOlApp.Explorers
For x = 1 To myOlExps.Count
   myOlExps.Item(x).WindowState = olMinimized
Next x
```

If you use VBScript, you do not create the **Application** object, and you cannot use named constants. This example shows how to perform the same task using VBScript.

```vba
For x = 1 To Application.Explorers.Count
   Application.Explorers.Item(x).WindowState = 2
Next
```
WordEditor Property

Returns the Microsoft Word Document Object Model of the message being displayed. This object model may be temporary and should not be stored for later use. The WordEditor property is only valid if IsWordMail returns True and the EditorType is olEditorWord. Read-only.

objInspector.WordEditor

objInspector Required. An expression that returns an Inspector object.
Example

The following VBScript example demonstrates the use of Word's object model. It adds a filled rectangle to the message area of the item.

Sub CommandButton1_Click()
    Set myInspector = Item.GetInspector
    Set WordDoc = myInspector.WordEditor
    Set Fill = WordDoc.Shapes.AddShape(1, 90, 90, 90, 50).Fill 'ms
    Fill.ForeColor.RGB = RGB(128, 0, 0)
    Fill.BackColor.RGB = RGB(170, 170, 170)
    Fill.TwoColorGradient 4, 1 'msoGradientDiagonalDown=4
End Sub
XML Property

Returns or sets a value that specifies the XML definition of the current view. The XML definition describes the view type by using a series of tags and keywords corresponding to various properties of the view itself. When the view is created, the XML definition is parsed to render the settings for the new view. Read/write String.

expression.XML

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

The following example creates a new view and displays its XML definition in the Immediate Window.

Sub XMLView()
'Creates a new view and displays the XML definition
'in the immediate window

    Dim olApp As Outlook.Application
    Dim objViews As Views
    Dim objView As View

    Set olApp = Outlook.Application
    Set objViews = _
    Set objView = objViews.Add _
        ("New Table View", olTableView, olViewSaveOptionAllFoldersOfT

    Debug.Print objView.XML

End Sub

The modified properties which are visible in the following XML source code. In addition to the property definitions, the XML source also defines any objects that make up the view. The following example displays the XML definition of columns that appear in the above view.

<column>
    <heading>Flag Status</heading>
    <prop>http://schemas.microsoft.com/mapi/proptag/0x10900003</prop>
    <type>i4</type>
    <bitmap>1</bitmap>
    <style>padding-left:3px;text-align:center;padding-left:3px</style>
</column>
<column>
    <format>boolicon</format>
    <heading>Attachment</heading>
    <prop>urn:schemas:httpmail:hasattachment</prop>
    <type>boolean</type>
YomiCompanyName Property

- Returns or sets a String indicating the Japanese phonetic rendering (yomigana) of the company name for the contact. Read/write.

objContactItem.YomiCompanyName

objContactItem  Required. An expression that returns a ContactItem object.
YomiFirstName Property

Returns or sets a String indicating the Japanese phonetic rendering (yomigana) of the first name for the contact. Read/write.

$objContactItem.YomiFirstName

$objContactItem Required. An expression that returns a ContactItem object.
YomiLastName Property

- Returns or sets a String indicating the Japanese phonetic rendering (yomigana) of the last name for the contact. Read/write.

`objContactItem.YomiLastName`

`objContactItem` Required. An expression that returns a `ContactItem` object.
Activate Event

Occurs when an explorer or inspector becomes the active window, either as a result of user action or through program code. This event is not available in VBScript.

Sub object_Activate()

object  An expression that evaluates to an Explorer or Inspector object.
Example

This code example uses the `WindowState` property to maximize the topmost explorer window when the `Activate` event occurs. The sample code must be placed in a class module, and the `Initialize_handler` routine must be called before the event procedure can be called by Microsoft Outlook.

```vba
Dim myOlApp As Outlook.Application
Public WithEvents myOlExp As Outlook.Explorer

Public Sub Initialize_handler()
    Set myOlExp = myOlApp.ActiveExplorer
End Sub

Private Sub myOlExp_Activate()
    If myOlExp.WindowsState = olNormalWindow Then _
        myOlExp.WindowsState = olMaximized
End Sub
```
AdvancedSearchComplete Event

Occurs when the AdvancedSearch method has completed. The AdvancedSearchComplete event is used to return the object that was created by the AdvancedSearch method.

Private Sub expression_ AdvancedSearchComplete(ByVal SearchObject As Object)

expression A variable which references an object of type Application declared with events in a class module.

SearchObject The Search object returned by the AdvancedSearch method.
Remarks

This event only fires when the **AdvancedSearch** method was executed programatically.
Example

The following example displays the number of objects returned by the search.

Private Sub olApp_AdvancedSearchComplete(ByVal SearchObject As Search)
    'Get Search object's results collection.
    Dim myResults As Results
    Set myResults = SearchObject.Results
    'Display the number of objects found in the search
    MsgBox "The number of objects found in the search is: " & my
End Sub
AdvancedSearchStopped Event

Occurs when a specified Search object's Stop method has been executed.

Private Sub application_AvancedSearchStopped(ByVal SearchObject As Object)

eexpression  A variable that references an object of type Application declared with events in a class module.

SearchObject  The Search object returned by the AdvancedSearch method.
Remarks

After this event is fired, the Search object’s Results collection will no longer be updated. This event can only be triggered programmatically.
Example

The following example displays the name of the Search object that was stopped.

Private Sub olApp_AdvancedSearchStopped(ByVal SearchObject As Search
    'Inform the user that the search has stopped.
    MsgBox "An AdvancedSearch has been stopped. " & _
        "The results that are returned are not necessarily the c
End Sub
AttachmentAdd Event

Occurs when an attachment has been added to an item.

Sub object_AttachmentAdd(Attachment As Attachment)

object An object that evaluates to one of the objects in the Applies To list. In VBScript, use the word Item.

Attachment Required. The Attachment that was added to the item.
Example

This VBScript example checks the size of the item after an embedded attachment has been added and displays a warning if the size exceeds 500,000 bytes.

Sub Item_AttachmentAdd(ByVal NewAttachment)
    If NewAttachment.Type = 1 Then
        Item.Save
        If Item.Size > 500000 Then
            MsgBox "Warning: Item size is now " & Item.Size & " byte"
        End If
    End If
End Sub
AttachmentRead Event

Occurs when an attachment in an item has been opened for reading.

Sub object_AttachmentRead(ByVal Attachment As Attachment)

object An object that evaluates to one of the objects in the Applies To list. In VBScript, use the word Item.

Attachment Required. The Attachment that was opened.
Example

This VBScript example reminds the user to also save changes to the original of an embedded file.

Sub Item_AttachmentRead(ByVal ReadAttachment)
    If ReadAttachment.Type = 1 then
        MsgBox "If you change this file, save your changes to the original as well."
    End If
End Sub
BeforeAttachmentSave Event

Occurs just before an attachment is saved.

Sub object_BeforeAttachmentSave(ByVal Attachment As Attachment, Cancel As Boolean)

object  An expression that evaluates to one of the objects in the Applies To list. In VBScript, use the word Item.

Attachment  Required. The Attachment to be saved.

Cancel  Optional (not used in VBScript). False when the event occurs. If the event procedure sets this argument to True, the save operation is not completed and the attachment is not changed.


Remarks

In VBScript, if you set the return value of this function to False, the save operation is cancelled and the attachment is not changed.
Example

This VBScript example notifies the user that the user is not allowed to save the attachment. It returns \textbf{False} to cancel the save operation.

Function Item\_BeforeAttachmentSave(ByVal SaveAttachment)
    MsgBox "You are not allowed to save " & SaveAttachment.Name
    Item\_BeforeAttachmentSave = False
End Function
BeforeCheckNames Event

Occurs just before Microsoft Outlook starts resolving names in the recipient collection for an item.

Sub object_BeforeCheckNames(Cancel As Boolean)

object An expression that evaluates to an object in the Applies To list. In VBScript, use the word Item.

Cancel Optional (not used in VBScript). False when the event occurs. If the event procedure sets this argument to True, the operation is cancelled and the names in the recipients collection are not resolved.
Remarks

In VBScript, if you set the return value of this function to **False**, the operation is cancelled and the names in the recipients collection are not resolved.
Example

This VBScript example asks the user if the user wants to resolve names and returns False to cancel the operation if the user answers no.

Function Item_BeforeCheckNames()
    If MsgBox ("Do you want to resolve names now?",4) = 7 Then
        Item_BeforeCheckNames = False
    End If
End Function
BeforeDelete Event

Occurs before an item is deleted.

Sub expression_BeforeDelete(ByVal Item As Object, Cancel As Boolean)

expression An object in the Applies To list declared with events in a class module.

Item Required Object. The item being deleted.

Cancel Required Boolean. False when the event occurs. If the event procedure sets this argument to True, the operation is not completed and the item is not deleted.
Remarks

In order for this event to fire when a mail message, distribution list, journal entry, task, contact or post are deleted through an action, an Inspector must be open.

The event occurs each time an item is deleted.
Example

The following example prompts the user regarding whether to delete the specified item. If the user clicks **No**, the item will not be deleted. If the user clicks **Yes**, the item will be deleted and a message will be displayed to the user.

Private Sub objMail_BeforeDelete(ByVal Item As Object, Cancel As Boolean)
'Prompts the user before deleting an item

    Dim strPrompt As String
    'Prompt the user for a response
    strPrompt = "Are you sure you want to delete the item?"
    If MsgBox(strPrompt, vbYesNo + vbQuestion) = vbNo Then
        'Don't delete the item
        Cancel = True
    Else
        MsgBox("Item Deleted")
        'Delete the item
        Cancel = False
    End If

End Sub
**BeforeFolderSwitch Event**

Occurs before the explorer navigates to a new folder, either as a result of user action or through program code. This event is not available in VBScript.

**Note**  If the folder being switched to is in a name space that doesn’t support Automation (such as the file system), *NewFolder* is *Nothing*.

**Sub** `object_BeforeFolderSwitch(ByVal NewFolder As Object, Cancel As Boolean)`

*object*  An expression that evaluates to an *Explorer* object.

*NewFolder*  Required. The *MAPIFolder* object the explorer is switching to.

*Cancel*  Optional. *False* when the event occurs. If the event procedure sets this argument to *True*, navigation is cancelled and the current folder is not changed.
Example

This sample prevents a user from switching to a folder named Off Limits. The sample code must be placed in a class module, and the Initialize_handler routine must be called before the event procedure can be called by Microsoft Outlook.

Dim myOlApp As Outlook.Application
Public WithEvents myOlExp As Outlook.Explorer

Public Sub Initialize_handler()
    Set myOlExp = myOlApp.ActiveExplorer
End Sub

Private Sub myOlExp_BeforeFolderSwitch(ByVal NewFolder As Object, Cancel As Boolean)
    If NewFolder.Name = "Off Limits" Then
        MsgBox "You do not have permission to access this folder."
        Cancel = True
    End If
End Sub
BeforeGroupAdd Event

Occurs before a new group is added to the Outlook Bar, either as a result of user action or through program code. This event is not available in VBScript.

Sub object_BeforeGroupAdd(Cancel As Boolean)

object  An expression that evaluates to an OutlookBarGroups collection object.

Cancel  Optional. False when the event occurs. If the event procedure sets this argument to True, the group is not added to the Outlook Bar.
Example

This example prevents the user from adding a group to the Outlook Bar. The sample code must be placed in a class module, and the Initialize_handler routine must be called before the event procedure can be called by Microsoft Outlook.

```vba
Dim myOlApp As New Outlook.Application
Dim WithEvents myOlGroups As Outlook.OutlookBarGroups
Dim myOlBar As Outlook.OutlookBarPane

Sub Initialize_handler()
    Set myOlBar = myOlApp.ActiveExplorer.Panes.Item("OutlookBar")
    Set myOlGroups = myOlBar.Contents.Groups
End Sub

Private Sub myOlGroups_BeforeGroupAdd(Cancel As Boolean)
    Cancel = True
End Sub
```
BeforeGroupRemove Event

Occurs before a new group is removed from the Outlook Bar, either as a result of user action or through program code. This event is not available in VBScript.

**Sub object** _BeforeGroupRemove(ByVal Group As OutlookBarGroup, Cancel As Boolean)_

*object* An expression that evaluates to an OutlookBarGroups collection object.

*Group* Required. The OutlookBarGroup that is being removed.

*Cancel* Optional. **False** when the event occurs. If the event procedure sets this argument to **True**, the group is not removed from the Outlook Bar.
Example

This example prevents the user from removing a group from the Outlook Bar. The sample code must be placed in a class module, and the Initialize_handler routine must be called before the event procedure can be called by Microsoft Outlook.

Dim myOlApp As New Outlook.Application
Dim WithEvents myOlGroups As Outlook.OutlookBarGroups
Dim myOlBar As Outlook.OutlookBarPane

Sub Initialize_handler()
    Set myOlBar = myOlApp.ActiveExplorer.Panes.item("OutlookBar")
    Set myOlGroups = myOlBar.Contents.Groups
End Sub

Private Sub myOlGroups_BeforeGroupRemove(ByVal Group As OutlookBarGroup, Cancel As Boolean)
    Cancel = True
End Sub
Show All
BeforeGroupSwitch Event

Occurs before a new group is opened in the Outlook Bar, either as a result of user action or through program code. This event is not available in VBScript.

Sub object_BeforeGroupSwitch(ByVal ToGroup As OutlookBarGroup, Cancel as Boolean)

object An expression that evaluates to an OutlookBarPane object.

ToGroup Required. The OutlookBarGroup being opened.

Cancel Optional. False when the event occurs. If the event procedure sets this argument to True, the change is cancelled and the current group remains open.
Example

This example prevents the user from opening the Outlook Bar group named Other Shortcuts. The sample code must be placed in a class module, and the Initialize_handler routine must be called before the event procedure can be called by Microsoft Outlook.

Dim myOlApp As New Outlook.Application
Public WithEvents myOlPane As Outlook.OutlookBarPane

Public Sub Initialize_handler()
    Set myOlPane = myOlApp.ActiveExplorer.Panes.Item("OutlookBar")
End Sub

Private Sub myOlPane_BeforeGroupSwitch(ByVal ToGroup As Outlook.OutlookBarGroup, Cancel As Boolean)
    If ToGroup.Name = "Other Shortcuts" Then Cancel = True
End Sub
BeforeItemCopy Event

Occurs when an item is copied. This event can be cancelled after it has started.

**Private Sub expression_BeforeItemCopy(Cancel As Boolean)**

*expression*  An expression that returns an [Explorer](#) object declared with events in a class module.

**Cancel**  Required [Boolean. False](#) when the event occurs. If the event procedure sets this argument to [True](#), the operation is not completed and the item is not copied.
Example

The following example prompts the user before an item is copied. A message is displayed to the user verifying that the item should be copied. If the user clicks Yes, the item is copied.

Private Sub objExplorer_BeforeItemCopy(Cancel As Boolean)
'Prompts the user before copying an item

    Dim lngAns As Long 'user answer
    'Display question to user
    lngAns = MsgBox("Are you sure you want to copy the item?", vbYesNo)
    If lngAns = vbYes Then
        Cancel = False
    Else
        'Set Cancel argument based on answer
        Cancel = True
    End If

End Sub
BeforeItemCut Event

Occurs when an item is cut from a folder. This method can be cancelled after it has started. If the event is cancelled, then the item will not be removed.

**Private** Sub explorer_BeforeItemCut(Cancel As Boolean)

*explorer*  An expression that returns an **Explorer** object.

*Cancel*  Optional. **False** when the event occurs. If the event procedure sets this argument to **True**, the operation is not completed and the item is not deleted.
Example

The following example prompts the user with a warning message before the item is cut from the folder. If the user clicks **Yes**, the item is cut from the folder. If the user clicks **No**, the item will not be removed from the folder.

Private Sub objExplorer_BeforeItemCut(Cancel As Boolean)
'prompts the user before cutting an item

    Dim lngAns As Long 'user answer
    'Display question to user
    lngAns = MsgBox("Are you sure you want to cut the item?", vbYesNo
    'Set cancel argument based on answer
    If lngAns = vbYes Then
        Cancel = False
    ElseIf lngAns = vbNo Then
        Cancel = True
    End If

End Sub
BeforeItemPaste Event

Occurs when a Microsoft Outlook item is pasted. This event can be cancelled after it has started.

Private Sub expression_BeforeItemPaste(ClipboardContent As Variant, ByVal Target As MAPIFolder, Cancel As Boolean)

expression  An expression that returns an Explorer object declared with events in a class module.

ClipboardContent  Required Variant. The content to be pasted.

Target  Required MAPIFolder. The destination of the paste.

Cancel  Required Boolean. False when the event occurs. If the event procedure sets this argument to True, the operation is not completed and the item is not deleted.
Example

The following example prompts the user before pasting the contents of the Clipboard to the specified target. If the user clicks **Yes**, the contents of the Clipboard are copied to the specified target destination and a message box is displayed for each item copied.

```vba
Private Sub objExplorer_BeforeItemPaste(ClipboardContent As Variant,

    Dim lngAns As Integer 'users' answer
    'Prompt user about paste
    lngAns = MsgBox("Are you sure you want to past the contents of the clipboard into the 
        & Target.Name & ", vbYesNo)

    If lngAns = vbYes Then
        'If user wants to paste
        If TypeOf ClipboardContent Is Selection Then
            'if is collection, display each object name
            Dim obj As Object
            For Each obj In ClipboardContent
                'Display subject of item
                MsgBox "Pastng Item: ", obj.Subject
            Next
        End If
        Cancel = False
    Else
        'If user clicks no, display message and cancel paste
        MsgBox "The clipboard content was not pasted."
        Cancel = True
    End If

End Sub
```
BeforeMaximize Event

Occurs when an Explorer or Inspector is maximized by the user. This event can be cancelled after it has started.

Private Sub expression_BeforeMaximize(Cancel As Boolean)

expression An expression that returns an object in the Applies To list declared with events in a class module.

Cancel Required Boolean. False when the event occurs. If the event procedure sets this argument to True, the operation is not completed and the explorer or Inspector is not maximized.
Example

The following example prompts the user with a warning message before maximizing the current window. If the user clicks Yes, the explorer will maximize.

Private Sub objExplorer_BeforeMaximize(Cancel As Boolean)
'Prompts the user before maximizing the explorer

    Dim lngAns As Long

    lngAns = MsgBox("Are you sure you want to maximize the current window?", vbYesNo)
    If lngAns = vbYes Then
        Cancel = False
    Else
        Cancel = True
    End If

End Sub
BeforeMinimize Event

Occurs when the active Explorer or Inspector is minimized by the user. This event can be cancelled after it has started.

**Private Sub** expression**_BeforeMinimize**(Cancel As Boolean)

ing An expression that returns one of the object in the Applies To list declared with events in a class module.

**Cancel** Required Boolean. *False* when the event occurs. If the event procedure sets this argument to *True*, the operation is not completed and the Explorer or Inspector is not minimized.
Example

The following example prompts the user with a message before the window is minimized. If the user clicks **Yes**, the Explorer is minimized.

Private Sub objExplorer_BeforeMinimize(Cancel As Boolean)
'Prompts the user before minimizing the Explorer

    Dim lngAns As Long

    lngAns = MsgBox("Are you sure you want to minimize the current w"
    If lngAns = vbYes Then
        Cancel = False
    Else
        Cancel = True
    End If
End If
End Sub
BeforeMove Event

Occurs when the Inspector or Explorer is moved by the user. This event can be cancelled after it has started.

**Private Sub** `expression_BeforeMove(Cancel As Boolean)`

`expression` An expression that returns an object in the Applies To list declared with events in a class module.

**Cancel** Required **Boolean. False** when the event occurs. If the event procedure sets this argument to **True**, the operation is not completed and the Explorer or Inspector is not moved.
Example

The following example prompts the user with a message before the **Explorer** is moved by the user. If the user clicks **Yes**, the **Explorer** can be moved by the user.

Private Sub objExplorer_BeforeMove(Cancel As Boolean)
'Prompts the user before moving the window

    Dim lngAns As Long

    lngAns = MsgBox("Are you sure you want to move the current windo
    If lngAns = vbYes Then
        Cancel = False
    Else
        Cancel = True
    End If

End Sub
BeforeNavigate Event

Occurs when the user clicks on an Outlook Bar shortcut to navigate to a different folder. This event is not available in VBScript.

Sub object_BeforeNavigate(ByVal Shortcut As OutlookBarShortcut, Cancel As Boolean)

object An expression that evaluates to an OutlookBarPane object.

Shortcut Required. The shortcut that the user clicked.

Cancel Optional. False when the event occurs. If the event procedure sets this argument to True, the current folder is not changed.
Example

This example prevents the user from using the Outlook Bar to open the Notes folder. The sample code must be placed in a class module, and the Initialize_handler routine must be called before the event procedure can be called by Microsoft Outlook.

Dim myOlApp As New Outlook.Application
Public WithEvents myOlPane As Outlook.OutlookBarPane

Public Sub Initialize_handler()
    Set myOlPane = myOlApp.ActiveExplorer.Panes.Item("OutlookBar")
End Sub
Private Sub myOlPane_BeforeNavigate(ByVal Shortcut As Outlook.OutlookBarShortcut, Cancel As Boolean)
    If Shortcut.Name = "Notes" Then
        MsgBox "You cannot view the Notes folder."
        Cancel = True
    End If
End Sub
BeforeReminderShow Event

Occurs before the reminder dialog is displayed.

Private Sub expression_BeforeReminderShow(Cancel As Boolean)

expression A variable which references an object of type Reminders declared with events in a class module.

Cancel Required. True to cancel the event. The default value is False.
**Example**

The following example prompts the user before displaying a reminder. The user can cancel the events by clicking **No**.

```vbnet
Private Sub objReminders_BeforeReminderShow(Cancel As Boolean)
    'Occurs before a reminder appears
    Dim lngAns As Long
    lngAns = MsgBox("Do you want to view the reminder?", vbYesNo)
    If lngAns = vbYes Then
        Cancel = False
    Else
        Cancel = True
    End If
End Sub
```
BeforeShortcutAdd Event

Occurs before a new shortcut is added to a group in the Outlook Bar, either as a result of user action or through program code. This event is not available in VBScript.

Sub object_BeforeShortcutAdd(Cancel As Boolean)

object An expression that evaluates to an OutlookBarShortcuts collection object.

Cancel Optional. False when the event occurs. If the event procedure sets this argument to True, the shortcut is not added to the group.
Example

The following example prevents a user from adding a shortcut to the first Outlook Bar group. The sample code must be placed in a class module, and the Initialize_handler routine must be called before the event procedure can be called by Microsoft Outlook.

Dim myOlApp As New Outlook.Application
Dim WithEvents myOlShortcuts As Outlook.OutlookBarShortcuts
Dim myOlBar As Outlook.OutlookBarPane

Sub Initialize_handler()
    Set myOlBar = myOlApp.ActiveExplorer.Panes.Item("OutlookBar")
    Set myOlShortcuts = myOlBar.Contents.Groups.Item(1).Shortcuts
End Sub

Private Sub myOlShortcuts_BeforeShortcutAdd(Cancel As Boolean)
    MsgBox "You are not allowed to add a shortcut to this group."
    Cancel = True
End Sub
BeforeShortcutRemove Event

Occurs before a new shortcut is removed from a group in the Outlook Bar, either as a result of user action or through program code. This event is not available in VBScript.

Sub object_BeforeShortcutRemove(ByVal Shortcut As OutlookBarShortcut, Cancel As Boolean)

object An expression that evaluates to an OutlookBarShortcuts collection object.

Shortcut Required. The OutlookBarShortcut that is being removed.

Cancel Optional. False when the event occurs. If the event procedure sets this argument to True, the shortcut is not removed from the group.
**Example**

The following example prevents a user from removing a shortcut from the first Outlook Bar group. The sample code must be placed in a class module, and the `Initialize_handler` routine must be called before the event procedure can be called by Microsoft Outlook.

```vbnet
Dim myOlApp As New Outlook.Application
Dim WithEvents myOlShortcuts As Outlook.OutlookBarShortcuts
Dim myOlBar As Outlook.OutlookBarPane

Sub Initialize_handler()
    Set myOlBar = myOlApp.ActiveExplorer.Panes.Item("OutlookBar")
    Set myOlShortcuts = myOlBar.Contents.Groups.Item(1).Shortcuts
End Sub

Private Sub myOlShortcuts_BeforeShortcutRemove(Cancel As Boolean)
    MsgBox "You are not allowed to remove a shortcut from this group"
    Cancel = True
End Sub
```
**BeforeSize Event**

Occurs when the user sizes the current **Explorer** or **Inspector**. This event can be cancelled after it has started. If the event is cancelled, the window is not sized.

**Private Sub** _expression_ **_BeforeSize**(Cancel As Boolean)

*expression*  An expression that returns an object in the Applies To list declared with events in a class module.

**Cancel**  Required **Boolean. False** when the event occurs. If the event procedure sets this argument to **True**, the operation is not completed and the **Explorer** or **Inspector** is not sized.
Example

The following example prompts the user with a warning message before the Inspector is sized. If the user clicks **Yes**, the Inspector can be sized.

Private Sub expression_BeforeSize(Cancel As Boolean)
'Prompts the user before resizing the window

    Dim lngAns As Long

    lngAns = MsgBox("Are you sure you want to resize the current win
    If lngAns = vbYes Then
        Cancel = False
    Else
        Cancel = True
    End If

End Sub
BeforeViewSwitch Event

Occurs before the explorer changes to a new view, either as a result of user action or through program code. This event is not available in VBScript.

Sub object_BeforeViewSwitch(ByVal NewView As String, Cancel As Boolean)

object An expression that evaluates to an Explorer object.

NewView Required. The name of the view the explorer is switching to.

Cancel Optional. False when the event occurs. If the event procedure sets this argument to True, the switch is cancelled and the current view is not changed.
Example

This example confirms that the user wants to switch views and cancels the switch if the user answers no. The sample code must be placed in a class module, and the Initialize_handler routine must be called before the event procedure can be called by Microsoft Outlook.

Dim myOlApp As Outlook.Application
Public WithEvents myOlExp As Outlook.Explorer

Public Sub Initialize_handler()
    Set myOlExp = myOlApp.ActiveExplorer
End Sub

Private Sub myOlExp_BeforeViewSwitch(ByVal NewView As Variant, Cancel As Boolean)
    Dim Prompt As String
    Prompt = "Are you sure you want to switch to the " & NewView & " view?"
    If MsgBox(Prompt, vbYesNo + vbQuestion) = vbNo Then Cancel = True
End Sub
Show All
Close Event

Occurs when the Inspector associated with an Outlook item is being closed.

Sub object_Close(Cancel As Boolean)

object An expression that evaluates to an object in the Applies To list. In VBScript, use the word Item.

Cancel Optional (not used in VBScript). False when the event occurs. If the event procedure sets this argument to True, the close operation is not completed and the inspector is left open.
Remarks

In VBScript, if you set the return value of this function to **False**, the close operation isn't completed and the inspector is left open. This event cannot be cancelled for the **Inspector** and **Explorer** objects.

If you use the **Close** method to fire this event, it can only be canceled if the Close method uses the **olPromptForSave** argument.
Example

This example tests for the **Close** event and if the **item** has not been **Saved**, it uses the **Save** method to save the item without prompting the user.

```vbnet
Function Item_Close()
    If Not Item.Saved Then
        Item.Save
    End If
End Function
```
CustomAction Event

Occurs when a custom action of an Outlook item executes. The Action object and the newly-created item resulting from the custom action are passed to the event.

Sub object_CustomAction(ByVal Action As Object, ByVal Response As Object, Cancel As Boolean)

object An expression that evaluates to one of the objects in the Applies To list. In VBScript, use the word Item.

Action Required. The Action object.

Response Required. The newly-created item resulting from the custom action.

Cancel Optional (not used in VBScript). False when the event occurs. If the event procedure sets this argument to True, the custom action is not completed.
Remarks

In VBScript, if you set the return value of this function to **False**, the custom action operation is not completed.
Example

This example uses the CustomAction event to set a property on the response item.

Function Item_CustonAction(ByVal myAction, ByVal myResponse)
    Select Case myAction.Name
        Case "Action1"
            myResponse.Subject = "Changed by VB Script"
        Case Else
    End Select
End Function
CustomPropertyChange Event

Occurs when a custom property of an Outlook item is changed. The property name is passed to the procedure so that you can determine which custom property changed.

Sub object_CustomPropertyChange(ByVal Name As String)

object An object that evaluates to one of the object in the Applies To list. In VBScript, use the word Item.

Name Required. The name of the custom property that was changed.
Example

This VBScript example uses the **CustomPropertyChange** event to enable a control when a Boolean field is set to **True**.

For this example, create two custom fields on the second page of a **form**. The first, a Boolean field, is named "RespondBy". The second field is named "DateToRespond".

```
Sub Item_CustomPropertyChange(ByVal myPropName)
    Select Case myPropName
        Case "RespondBy"
            Set myPages = Item.GetInspector.ModifiedFormPages
            Set myCtrl = myPages("P.2").Controls("DateToRespond")
            If Item.UserProperties("RespondBy").Value Then
                myCtrl.Enabled = True
                myCtrl.Backcolor = 65535 'Yellow
            Else
                myCtrl.Enabled = False
                myCtrl.Backcolor = 0 'Black
            End If
        Case Else
    End Select
End Sub
```
Deactivate Event

Occurs when an explorer or inspector ceases to be the active window, either as a result of user action or through program code. This event is not available in VBScript.

Sub object_Deactivate()

object  An expression that evaluates to an Explorer or Inspector object.
**Example**

This example uses the `WindowState` property to minimize the topmost explorer window when it is not active. The sample code must be placed in a class module, and the `Initialize_handler` routine must be called before the event procedure can be called by Microsoft Outlook.

```vba
Dim myOlApp As Outlook.Application
Public WithEvents myOlExp As Outlook.Explorer

Public Sub Initialize_handler()
    Set myOlExp = myOlApp.ActiveExplorer
End Sub

Private Sub myOlExp_Deactivate()
    myOlExp.WindowState = olMinimized
End Sub
```
FolderAdd Event

Occurs when a folder is added to the specified Folders collection. This event is not available in VBScript.

Sub object_FolderAdd(ByVal Folder As MAPIFolder)

object An expression that evaluates to a Folders collection object.

Folder Required. The MAPIFolder that was added to the collection.
Example

This example displays a new folder created in the user’s Inbox folder.

Dim myOlApp As New Outlook.Application
Public WithEvents myOlFolders As Outlook.Folders

Public Sub Initialize_handler()
    Set myOlFolders = myOlApp.GetNamespace("MAPI").GetDefaultFolder()
End Sub

Private Sub myOlFolders_FolderAdd(ByVal Folder As Outlook.MAPIFolder)
    Folder.Display
End Sub
FolderChange Event

Occurs when a folder in the specified Folders collection is changed. This event is not available in VBScript.

Sub object_FolderChange(ByVal Folder As MAPIFolder)

object An expression that evaluates to a Folders collection object.

Folder Required. The MAPIFolder that was changed.
Example

This example prompts the user to remove a folder from the Deleted Items folder if the folder is empty. The sample code must be placed in a class module, and the Initialize_handler routine must be called before the event procedure can be called by Microsoft Outlook.

Dim myolapp As New Outlook.Application
Dim WithEvents myFolders As Outlook.Folders

Sub Initialize_handler()
    Set myNS = myolapp.GetNamespace("MAPI")
    Set myFolders = myNS.GetDefaultFolder(olFolderDeletedItems).Folders
End Sub

Private Sub myFolders_FolderChange(ByVal Folder As Outlook.MAPIFolder)
    If Folder.Items.Count = 0 Then
        MyPrompt = Folder.Name & " is empty. Do you want to delete it?"
        If MsgBox(MyPrompt, vbYesNo + vbQuestion) = vbYes Then
            Folder.Delete
        End If
    End If
End If
End Sub
FolderRemove Event

Occurs when a folder is removed from the specified Folders collection. This event is not available in VBScript.

Sub object_FolderRemove()

object   An expression that evaluates to a Folders collection object.
Example

This example fills a combo box on a form with the names of the folders in the Deleted Items folder. The sample code must be placed in a class module, and the Initialize_handler routine must be called before the event procedure can be called by Microsoft Outlook.

Dim myolapp As New Outlook.Application
Dim WithEvents myFolders As Outlook.Folders

Sub Initialize_handler()
    Set myNS = myolapp.GetNamespace("MAPI")
    Set myFolders = myNS.GetDefaultFolder(olFolderDeletedItems).Folders
End Sub

Private Sub myFolders_FolderRemove()
    Form1.Combo1.Clear
    For x = 1 To myFolders.Count
        Form1.Combo1.AddItem (myFolders.Item(x).Name)
    Next x
End Sub
FolderSwitch Event

Occurs when the explorer navigates to a new folder, either as a result of user action or through program code. This event is not available in VBScript.

Sub object_FolderSwitch()

object  An expression that evaluates to an Explorer object.
Example

This example shows or hides a menu bar, depending on the folder that is visible. The sample code must be placed in a class module, and the Initialize_handler routine must be called before the event procedure can be called by Microsoft Outlook.

```vbs
Dim myOlApp As Outlook.Application
Public WithEvents myOlExp As Outlook.Explorer

Public Sub Initialize_handler()
    Set myOlExp = myOlApp.ActiveExplorer
End Sub

Private Sub myOlExp_FolderSwitch()
    Select Case myOlExp.CurrentFolder.Name
    Case "Sales Contacts"
        MyToolsMenu.Visible = True
    Case Else
        MyToolsMenu.Visible = False
    End Select
End Sub
```
Forward Event

Occurs when the user selects the **Forward** action for an Outlook **item**.

**Sub object** _Forward(ByVal Forward As Object, Cancel As Boolean)_

*object*  An expression that evaluates to one of the objects in the Appliees To list. In VBScript, use the word **Item**.

**Forward**  The new item being forwarded.

**Cancel**  Optional (not used in VBScript). **False** when the event occurs. If the event procedure sets this argument to **True**, the forward operation is not completed and the new item is not displayed.
Remarks

In VBScript, if you set the return value of this function to False, the forward action is not completed and the new item is not displayed.
Example

This example uses the **Forward** event to disable forwarding on an **item** by setting the return value to **False** and displays a message that the item may not be forwarded.

```vba
Function Item_Forward(ByVal myForwardItem)
    MsgBox "You may not forward this message!"
    ItemForward = False
End Function
```
**GroupAdd Event**

Occurs when a new group has been added to the Outlook Bar. This event is not available in VBScript.

**Sub object_GroupAdd(ByVal NewGroup As OutlookBarGroup)**

*object*  An expression that evaluates to an OutlookBarGroups object.

*NewGroup*  Required. The OutlookBarGroup that was added.
Example

This example adds a shortcut to a group when the group is created. The sample code must be placed in a class module, and the Initialize_handler routine must be called before the event procedure can be called by Microsoft Outlook.

Dim myOlApp As New Outlook.Application
Dim WithEvents myOlGroups As Outlook.OutlookBarGroups
Dim myOlBar As Outlook.OutlookBarPane

Sub Initialize_handler()
    Set myOlBar = myOlApp.ActiveExplorer.Panes.Item("OutlookBar")
    Set myOlGroups = myOlBar.Contents.Groups
End Sub

Private Sub myOlGroups_GroupAdd(ByVal NewGroup As Outlook.OutlookBarGroup)
    Dim myFolder As Outlook.MAPIFolder
    Set myFolder = myOlApp.GetNamespace("MAPI").GetDefaultFolder(olF
    NewGroup.Shortcuts.Add myFolder, "Calendar"
End Sub
ItemAdd Event

Occurs when an item is added to the specified collection. This event is not available in VBScript.

Sub object_ItemAdd(ByVal Item As Object)

object An expression that evaluates to one of the objects in the Applies To list.

Item Required. The item that was added.
Example

In this example, when a new contact is added to the Contacts folder, the contact item is attached to a mail message and sent to a distribution list named Sales Team. The sample code must be placed in a class module, and the Initialize_handler routine must be called before the event procedure can be called by Microsoft Outlook.

Dim myOlApp As Outlook.Application
Public WithEvents myOlItems As Outlook.Items

Public Sub Initialize_handler()
    Set myOlItems = myOlApp.GetNamespace("MAPI").GetDefaultFolder(olFolderContacts).Items
End Sub

Private Sub myOlItems_ItemAdd(ByVal Item As Object)
    Dim myOlMItem As Outlook.MailItem
    Dim myOlAtts As Outlook.Attachments
    Set myOlMItem = myOlApp.CreateItem(olMailItem)
    myOlMItem.Save
    Set myOlAtts = myOlMItem.Attachments
    ' Add new contact to attachments in mail message
    myOlAtts.Add Item, olByValue
    myOlMItem.To = "Sales Team"
    myOlMItem.Subject = "New contact"
    myOlMItem.Send
End Sub
**ItemChange Event**

Occurs when an item in the specified collection is changed. This event is not available in VBScript.

```vbnet
Sub object_ItemChange(ByVal Item As Object)

object  An expression that evaluates to one of the objects in the Applies To list.

Item   Required. The item that was changed.
```
Example

This example uses the `Start` property of the `AppointmentItem` object to determine if the appointment starts after normal business hours. If it does, and if the `Sensitivity` property of the `AppointmentItem` object is not already set to `olPrivate`, the example offers to mark the appointment as private.

```vba
Dim myOlApp As New Outlook.Application
Public WithEvents myOlItems As Outlook.Items
Public Sub Initialize_handler()
    Set myOlItems = myOlApp.GetNamespace("MAPI").GetDefaultFolder(olFolderCalendar).Items
End Sub

Private Sub myOlItems_ItemChange(ByVal Item As Object)
    If Format(Item.Start, "h") >= "17" And Item.Sensitivity <> olPrivate Then
        Prompt = "Appointment occurs after hours. Mark it private?"
        If MsgBox(Prompt, vbYesNo + vbQuestion) = vbYes Then
            Item.Sensitivity = olPrivate
            Item.Display
        End If
    End If
End Sub
```

ItemRemove Event

Occurs when an item is deleted from the specified collection. This event is not available in VBScript.

Sub object_ItemRemove()

object  An expression that evaluates to one of the objects in the Applies To list.
Example

This example optionally sends a notification message to a workgroup when the user removes a contact from the default Contacts folder. The sample code must be placed in a class module, and the Initialize_handler routine must be called before the event procedure can be called by Microsoft Outlook.

```
Dim myOlApp As Outlook.Application
Public WithEvents myOlItems As Outlook.Items

Public Sub Initialize_handler()
    Set myOlItems = myOlApp.GetNamespace("MAPI").GetDefaultFolder(olFolderContacts).Items
End Sub

Private Sub myOlItems_ ItemRemove()
    Dim myOlMItem As Outlook.MailItem
    If MsgBox("Do you want to notify the Sales Team?", vbYesNo + vbQ) = vbYes Then
        Set myOlMItem = myOlApp.CreateItem(olMailItem)
        myOlMItem.To = "Sales Team"
        myOlMItem.Subject = "Remove Contact"
        myOlMItem.Body = "Please remove the following contact from your list:"
        myOlMItem.Display
    End If
End Sub
```
ItemSend Event

- Occurs whenever an item is sent, either by the user through an Inspector (before the inspector is closed, but after the user clicks the Send button) or when the Send method is used in a program. This event is not available in VBScript.

Sub object_ItemSend(ByVal Item As Object, Cancel As Boolean)

object An expression that evaluates to an Application object.

Item Required. The item being sent.

Cancel Optional. False when the event occurs. If the event procedure sets this argument to True, the send action is not completed and the inspector is left open.
Example

The following example shows how to cancel the ItemSend event in response to user input.

Private Sub myOlApp_ItemSend(ByVal Item As Object, Cancel As Boolean)
    Prompt$ = "Are you sure you want to send " & Item.Subject & "?"
    If MsgBox(Prompt$, vbYesNo + vbQuestion, "Sample") = vbNo Then
        Cancel = True
    End If
End Sub
MapiLogonComplete Event

Occurs after the user has logged onto the system.

Private Sub expression_MapiLogonComplete()

expression A variable which references an object of type Application declared with events in a class module.
Example

The following example displays a message after the user has logged onto the system.

Private Sub objApp_MapiLogonComplete()
'Occurs when a user has logged on

    MsgBox "Logon complete"

End Sub
NewExplorer Event

Occurs whenever a new explorer window is opened, either as a result of user action or through program code. This event is not available in VBScript.

Sub object_NewExplorer(ByVal Explorer As Explorer)

object An expression that evaluates to an Explorers collection object.

Explorer Required. The explorer that was opened.
Remarks

The event occurs after the new Explorer object is created but before the explorer window appears.
Example

This example minimizes the currently active explorer window when a new explorer is about to appear. The sample code must be placed in a class module, and the Initialize_handler routine must be called before the event procedure can be called by Microsoft Outlook.

```vba
Dim myOlApp As New Outlook.Application
Public WithEvents myOlExplorers As Outlook.Explorers

Public Sub Initialize_handler()
    Set myOlExplorers = myOlApp.Explorers
End Sub

Private Sub myOlExplorers_NewExplorer(ByVal Explorer As Outlook.Explorer)
    If TypeName(myOlApp.ActiveExplorer) <> "Nothing" Then
        myOlApp.ActiveExplorer.WindowState = olMinimized
    End If
End Sub
```
NewInspector Event

Occurs whenever a new inspector window is opened, either as a result of user action or through program code. This event is not available in VBScript.

Sub object_NewInspector(ByVal Inspector As Inspector)

object An expression that evaluates to an Inspectors collection object.

Inspector Required. The inspector that was opened.
Remarks

The event occurs after the new **Inspector** object is created but before the inspector window appears.
Example

This example displays the Standard and Formatting toolbars of an inspector when it is opened. The sample code must be placed in a class module, and the Initialize_handler routine must be called before the event procedure can be called by Microsoft Outlook.

```vba
Dim myOlApp As New Outlook.Application
Public WithEvents myOlInspectors As Outlook.Inspectors

Public Sub Initialize_handler()
    Set myOlInspectors = myOlApp.Inspectors
End Sub

Private Sub myOlInspectors_NewInspector(ByVal Inspector As Outlook.Inspector)
    Inspector.CommandBars.Item("Formatting").Visible = True
End Sub
```
NewMail Event

Occurs when one or more new messages are received in the Inbox. This event is not available in VBScript.

Sub object_NewMail()

object  An expression that evaluates to an Application object.
Example

This example displays the Inbox folder when new mail arrives. The sample code must be placed in a class module, and the Initialize_handler routine must be called before the event procedure can be called by Microsoft Outlook.

Dim WithEvents myOlApp As Outlook.Application

Sub Initialize_handler()
    Set myOlApp = CreateObject("Outlook.application")
End Sub

Private Sub myOlApp_NewMail()
    Dim myExplorers As Outlook.Explorers
    Dim myFolder As Outlook.MAPIFolder
    Set myExplorers = myOlApp.Explorers
    Set myFolder = myOlApp.GetNamespace("MAPI").GetDefaultFolder(olFolderInbox)
    If myExplorers.Count <> 0 Then
        For x = 1 To myExplorers.Count
            On Error GoTo skipif
            If myExplorers.Item(x).CurrentFolder.Name = "Inbox" Then
                myExplorers.Item(x).Display
                myExplorers.Item(x).Activate
                Exit Sub
            End If
        skipif:
        Next x
    End If
    myFolder.Display
End Sub
OnError Event

Occurs when Microsoft Outlook encounters an error while synchronizing a user’s folders using the specified synchronization profile. This event is not available in VBScript.

Sub object_OnError(ByVal Code As Long, ByVal Description As String)

object  An expression that evaluates to a SyncObject object.

Code  A unique value that identifies the error.

Description  Required. A textual description of the error.
Example

This example displays a message box describing the synchronization error and sets attributes of controls on a form. The sample code must be placed in a class module, and the Initialize_handler routine must be called before the event procedure can be called by Microsoft Outlook.

Dim myOlApp As New Outlook.Application
Dim WithEvents mySync As Outlook.SyncObject
Dim myForm As New Form1

Sub Initialize_handler()
    Set mySync = myOlApp.Session.SyncObjects.Item(1)
End Sub

Private Sub mySync_OnError(ByVal Code As Long, ByVal Description As String)
    myForm.Label1.Caption = "Synchronization failed."
    mySync.Stop
    myForm.Command1.Enabled = False
    myForm.Command2.Enabled = False
    MsgBox "Unexpected sync error" & Str(Code) & ": " & Description
End Sub
Open Event

Occurs when a Microsoft Outlook item is being opened in an Inspector. When this event occurs, the Inspector object is initialized but not yet displayed. The Open event differs from the Read event in that Read occurs whenever the user selects the item in a view that supports in-cell editing as well as when the item is being opened in an Inspector.

Sub object_Open(Cancel As Boolean)

object An expression that evaluates to one of the objects in the Applies To list. In VBScript, use the word Item.

Cancel Optional (not used in VBScript). False when the event occurs. If the event procedure sets this argument to True, the open operation is not completed and the inspector is not displayed.
Remarks

In VBScript, if you set the return value of this function to **False**, the open operation is not completed and the inspector is not displayed.
Example

This VBScript example uses the **Open** event to display the "All Fields" page every time the **item** is opened.

Function **Item_Open()**
   Item.GetInspector.SetCurrentFormPage "All Fields"
End Function

This VBScript example uses the **Unread** property to detect whether the item has been previously read. If it has, then it asks if the user wants to open it. If the user answers No, the return value is set to **False** to prevent the item from opening.

Function **Item_Open()**
   If Item.UnRead = False Then
      myMsg = "Do you want to open this message again?"
      If MsgBox(myMsg, 4) = 6 Then
         Item_Open = True
      Else
         Item_Open = False
      End If
   End If
End Function
OptionsPagesAdd Event

Occurs whenever the Options dialog box (available through the Tools menu) or a folder Properties dialog box is opened. This event is not available in VBScript.

Sub object_OptionsPagesAdd(ByVal Pages As PropertyPages, ByVal Folder As MAPIFolder)

object  An expression that evaluates to an Application or a NameSpace object.

Pages  Required. The collection of property pages that have been added to the dialog box. This collection includes only custom property pages. It does not include standard Microsoft Outlook property pages.

Folder  This argument is only used with the MAPIFolder object. Required. The MAPIFolder object for which the Properties dialog box is being opened.
Remarks

Your program handles this event to add a custom property page. If `object` is an `Application` object, the property page will be added to the `Options` dialog box. If `object` is a `NameSpace` object, the property page will be added to `Properties` dialog box of the specified folder. When the event fires, the `PropertyPages` collection object identified by `Pages` contains the property pages that have been added prior to the event handler being called. To add your property page to the collection, use the `Add` method of the `PropertyPages` collection before exiting the event handler.
Example

This example adds a new property page to the Microsoft Outlook Options dialog box. The sample code must be placed in a class module of a Common Object Model (COM) add-in.

Implements IDTExtensibility2
Private WithEvents OutlApp As Outlook.Application

Private Sub IDTExtensibility2_OnConnection(ByVal Application As Object, ByVal ConnectMode As AddInDesignerObjects.ext_ConnectMode, ByVal AddInInst As Object, custom() As Variant)
  Set OutlApp = Application
End Sub

Private Sub OutlApp_OptionsPagesAdd(ByVal Pages As Outlook.PropertyPages)
  Dim myPage As Object
  Set myPage = CreateObject("PPE.CustomButton")
  Pages.Add myPage
End Sub
Progress Event

Occurs periodically while Microsoft Outlook is synchronizing a user’s folders using the specified synchronization profile. This event is not available in VBScript.

Sub object_Progress(Val State As OlSyncState, ByVal Description As String, ByVal Value As Long, ByVal Max As Long)

object  An expression that evaluates to a SyncObject object.

State  Required. A value that identifies the current state of the synchronization process. Can be either of the following OlSyncState constants: olSyncStarted or olSyncStopped.

Description  Required. A textual description of the current state of the synchronization process.

Value  Required. Specifies the current value of the synchronization process (such as the number of items synchronized).

Max  Required. The maximum that Value can reach. The ratio of Value to Max represents the percent complete of the synchronization process.
Example

This example updates a label on a form to show the progress of synchronization. The sample code must be placed in a class module, and the `Initialize_handler` routine must be called before the event procedure can be called by Microsoft Outlook.

```vba
Dim myOlApp As New Outlook.Application
Dim WithEvents mySync As Outlook.SyncObject
Dim myForm As New Form1

Sub Initialize_handler()
    Set mySync = myOlApp.Session.SyncObjects.Item(1)
End Sub

Private Sub mySync_Progress(ByVal State As Outlook.olSyncState, ByVal Description As String, ByVal Value As Long, ByVal Max As Long)
    If State = olSyncStarted Then
        Cap = "Synchronization started: "
    Else
        Cap = "Synchronization stopped: "
    End If
    Cap = Cap & Str(State / Max * 100) & "% " & Description
    Form1.Label1.Caption = Cap
End Sub
```
Show All
PropertyChange Event

Occurs when a standard property (for example, Subject or To) of a Microsoft Outlook item is changed. The property name is passed to the event so that you can determine which property was changed.

Sub object_PropertyChange(ByVal Name As String)

object One of the objects in the Applies To list. In VBScript, use the word Item.

Name Required. The name of the property that was changed.
Example

This example uses the **PropertyChange** event to disable the setting of a reminder on an **item**.

```vba
Sub Item_PropertyChange(ByVal myPropertyName)
    Select Case myPropertyName
        Case "ReminderSet"
            MsgBox "You may not set a reminder on this item!"
            Item.ReminderSet = False
        Case Else
    End Select
End Sub
```
Quit Event

Occurs when Microsoft Outlook begins to close. This event is not available in VBScript.

Sub object_Quit()

object  An expression that evaluates to an Application object.
Example

This example displays a farewell message when Microsoft Outlook exits. The sample code must be placed in a class module, and the Initialize_handler routine must be called before the event procedure can be called by Outlook.

Dim WithEvents myOlApp As Outlook.Application

Sub Initialize_handler()
    Set myOlApp = CreateObject("Outlook.application")
End Sub

Private Sub myOlApp_Quit()
    MsgBox "Goodbye, " & Application.GetNamespace("MAPI").CurrentUser
End Sub
Read Event

Occurs when an existing Microsoft Outlook item is opened for editing by the user. The Read event differs from the Open event in that Read occurs whenever the user selects the item in a view that supports in-cell editing as well as when the item is being opened in an Inspector.

Sub object_Read()

object

An object that evaluates to one of the objects in the Applies To list. In VBScript, use the word Item.
Example

This VBScript example uses the **Read** event to increment a counter that tracks how often an **item** is read.

```vbnet
Sub Item_Read()
    Set myProperty = Item.UserProperties("ReadCount")
    myProperty.Value = myProperty.Value + 1
    Item.Save
End Sub
```
Reminder Event

Occurs immediately before a reminder is displayed.

Sub object_Reminder(ByVal Item As Object)

object  An expression that evaluates to an Application object.

Item  The AppointmentItem, MailItem, ContactItem or TaskItem associated with the reminder. If the appointment associated with the reminder is a recurring appointment, Item is the specific occurrence of the appointment that displayed the reminder, not the master appointment.
Example

This Microsoft Visual Basic/Visual Basic for Applications example tests the item generating the reminder to determine if it is a mail item. If it is, the example uses the `ReplyAll` method to create and display a new mail item. The sample code must be placed in a class module, and the `Initialize_handler` routine must be called before the event procedure can be called by Microsoft Outlook.

```
Dim WithEvents myolapp As Outlook.Application

Sub Initialize_handler()
    Set myolapp = CreateObject("Outlook.application")
End Sub

Private Sub myolapp_Reminder(ByVal Item As Object)
    Dim myReplyItem As Outlook.MailItem
    If TypeName(Item) = "MailItem" Then
        Set myReplyItem = Item.ReplyAll
        myReplyItem.Display
    End If
End Sub
```
ReminderAdd Event

Occurs after a reminder is added.

Private Sub expression_ReminderAdd(ByVal ReminderObject As Reminder)

eexpression  A variable which references an object of type Reminders declared with events in a class module.

ReminderObject  Required. The Reminder object added to the collection.
Remarks

A reminder is not actually created until the associated Microsoft Outlook item has been saved. Therefore, this event will not occur until the associated item object has been saved.
Example

The following example displays the name of the new reminder when a reminder is added to the collection.

Private Sub objReminders_ReminderAdd(ByVal ReminderObject As Reminder)
    ' Occurs when a Reminder object is added to the collection
    MsgBox "The following reminder has been added: " & _
    ReminderObject.Caption
End Sub
ReminderChange Event

Occurs after a reminder has been modified.

**Private Sub** `expression_ReminderChange(ByVal ReminderObject As Reminder)`

`expression`  A variable which references an object of type `Reminders` declared with events in a class module.

**ReminderObject**  Required. The `Reminder` object that has been modified.
Example

The following example prompts the user with a message every time a reminder is modified.

Private Sub objReminders_ReminderChange(ByVal ReminderObject As Reminder)
' Occurs when reminder is changed

    MsgBox "The reminder " & ReminderObject.Caption & " has changed.";

End Sub
ReminderFire Event

Occurs before the reminder is executed.

**Private Sub** *expression* _ReminderFire*(ByVal *ReminderObject* As Reminder)*

*expression* A variable which references an object of type **Reminders** declared with events in a class module.

*ReminderObject* Required. The **Reminder** object that has been executed.
Example

The following example displays a message to the user every time a reminder is executed.

Private Sub objReminders_ReminderFire(ByVal ReminderObject As Reminder)
' Occurs when a reminder executes
    MsgBox "The reminder " & ReminderObject.Caption & " has just exe
End Sub
ReminderRemove Event

Occurs when a Reminder object has been removed from the collection.

Private Sub expression_ReminderRemove( )

expression  A variable which references an object of type Reminders declared with events in a class module.
Remarks

A reminder can be removed from the **Reminders** collection by any of the following means:

- The **Reminders** collection's **Remove** method.
- The **Reminder** object's **Dismiss** method.
- When the user clicks the **Dismiss** button.
- When a user turns off a meeting reminder from within the associated item.
- When a user deletes an item that contains a reminder.
Example

The following example displays a message to the user when a Reminder object is removed from the collection.

Private Sub objReminders_ReminderRemove()
'Occurs when a reminder is removed from the collection
'or the user clicks Dismiss

    MsgBox "A reminder has been removed from the collection."

End Sub
Reply Event

Occurs when the user selects the **Reply** action for an Outlook **item**.

**Sub object_Reply(ByVal Response As Object, Cancel As Boolean)**

*object*  An expression that evaluates to one of the objects in the Applies To list. In VBScript, use the word **Item**.

*Response*  The new item being sent in response to the original message.

*Cancel*  Optional (not used in VBScript). **False** when the event occurs. If the event procedure sets this argument to **True**, the reply operation is not completed and the new item is not displayed.
Remarks

In VBScript, if you set the return value of this function to False, the reply action is not completed and the new item is not displayed.
Example

This VBScript example uses the **Reply** event and sets the Sent Items folder for the reply **item** to the folder in which the original item resides.

```vbscript
Function Item_Replay(ByVal myResponse)
    Set myResponse.SaveSentMessageFolder = Item.Parent
End Function
```
ReplyAll Event

Occurs when the user selects the ReplyAll action for an Outlook item.

Sub object_ReplyAll(ByVal Response As Object, Cancel As Boolean)

object  An expression that evaluates to one of the objects in the Applies To list.

Response  The new item being sent in response to the original message.

Cancel  Optional. False when the event occurs. If the event procedure sets this argument to True, the reply all operation is not completed and the new item is not displayed.
**Example**

This VBScript example uses the **ReplyAll** event and reminds the user that proceeding will reply to all original recipients of an **item** and, depending on the user's response, either allows the action to continue or stops it.

```vbnet
Function Item_ReplyAll(ByVal myResponse)
    myMsg = "Do you really want to reply to all original recipients?"
    myResult = MsgBox(myMsg, 289, "Flame Protector")
    If myResult = 1 Then
        Item_ReplyAll = True
    Else
        Item_ReplyAll = False
    End If
End Function
```
SelectionChange Event

Occurs when the selection of the current view changes. Other selection changes (such as the selected folder) do not cause this event to occur. In addition, this event does not occur if the current folder is a file-system folder or if Outlook Today or any folder with a current Web view is displayed. This event is not available in VBScript.

Sub object_SelectionChange()

object An expression that evaluates to an Explorer object.
Example

This example changes the caption of a form named Form1 to show the number of items selected in the topmost explorer window. The sample code must be placed in a class module, and the Initialize_handler routine must be called before the event procedure can be called by Microsoft Outlook.

Dim myOlApp As Outlook.Application
Public WithEvents myOlExp As Outlook.Explorer

Public Sub Initialize_handler()
    Set myOlExp = myOlApp.ActiveExplorer
End Sub

Private Sub myOlExp_SelectionChange()
    Form1.Caption = myOlExp.Selection.Count & " items selected."
End Sub
Send Event

Occurs when the user selects the Send action for a Microsoft Outlook item.

Sub object_Send(Cancel As Boolean)

object An expression that evaluates to one of the objects in the Applies To list. In VBScript, use the word Item.

Cancel Optional (not used in VBScript). False when the event occurs. If the event procedure sets this argument to True, the send operation is not completed and the inspector is left open.
Remarks

In VBScript, if you set the return value of this function to **False**, the item is not sent.
Example

This VBScript example uses the **Send** event and sends an **item** with an automatic expiration date of one week.

```vbscript
Function Item_Send()
    Item.ExpiryTime = Date + 7
End Function
```
Show All
ShortcutAdd Event

Occurs when a new shortcut is added to a Outlook Bar group. This event is not available in VBScript.

Sub object ShortcutAdd(ByVal NewShortcut As OutlookBarShortcut)

object An expression that evaluates to an OutlookBarShortcuts collection object.

NewShortcut Required OutlookBarShortcut object. The shortcut that is being added.
Example

This example changes the name of a Calendar shortcut when it is added to the first group in the Outlook Bar. The sample code must be placed in a class module, and the Initialize_handler routine must be called before the event procedure can be called by Microsoft Outlook.

Dim myOlApp As New Outlook.Application
Dim WithEvents myOlSCuts As Outlook.OutlookBarShortcuts
Dim myOlBar As Outlook.OutlookBarPane

Sub Initialize_handler()
    Set myOlBar = myOlApp.ActiveExplorer.Panes.Item("OutlookBar")
End Sub

Private Sub myOlSCuts_ShortcutAdd(ByVal NewShortcut As Outlook.OutlookBarShortcut)
    Dim myNS As Outlook.NameSpace
    Set myNS = myOlApp.GetNamespace("MAPI")
    If NewShortcut.Target.Name = "Calendar" Then
        NewShortcut.Name = myNS.CurrentUser & "'s Schedule"
    End If
End Sub
Snooze Event

Occurs when a reminder is dismissed using the snooze button.

**Private Sub** `expression_Snooze(ByVal ReminderObject As Reminder)`

`expression` A variable which references an object of type **Reminders** declared with events in a class module.
Remarks

This event will fire when the `Snooze` method is executed, or when the user clicks the `Snooze` button.
Example

The following example displays the caption of the **Reminder** object that has been snoozed.

Private Sub expression_Snooze(ByVal ReminderObject As Reminder)
' Occurs when a user clicks snooze or when snooze is ' programmatically executed

    MsgBox "The reminder " & ReminderObject.Caption & " has been dismissed by using Snooze."

End Sub
Startup Event

Occurs when Microsoft Outlook is starting, but after all add-in programs have been loaded. This event is not available in VBScript.

Sub object_Startup()

object  An expression that evaluates to an Application object.
Remarks

An Outlook Visual Basic for Applications macro can use this event procedure to initialize itself when Outlook starts.
Example

This Microsoft Outlook Visual Basic for Applications example maximizes the Outlook explorer window when Outlook starts.

Private Sub Application_\texttt{Startup}()  
    Application.ActiveExplorer.WindowState = olMaximized  
End Sub
SyncEnd Event

Occurs immediately after Microsoft Outlook finishes synchronizing a user’s folders using the specified synchronization profile. This event is not available in VBScript.

Sub object_SyncEnd()

object An expression that evaluates to a SyncObject object.
Example

This example updates a label on a form to indicate that synchronization has finished and changes the enabled state of buttons on the form. The sample code must be placed in a class module, and the Initialize_handler routine must be called before the event procedure can be called by Microsoft Outlook.

Dim myOlApp As New Outlook.Application
Dim WithEvents mySync As Outlook.SyncObject
Dim myForm As New Form1

Sub Initialize_handler()
    Set mySync = myOlApp.Session.SyncObjects.Item(1)
End Sub

Private Sub mySync_SyncEnd()
    Form1.Label1.Caption = "Synchronization complete."
    Form1.Command1.Enabled = True
    Form1.Command2.Enabled = False
End Sub
SyncStart Event

Occurs when Microsoft Outlook begins synchronizing a user’s folders using the specified synchronization profile. This event is not available in VBScript.

Sub object_SyncStart()

object  An expression that evaluates to a SyncObject object.
Example

This example updates a label on a form to show that synchronization has started. The sample code must be placed in a class module, and the Initialize_handler routine must be called before the event procedure can be called by Microsoft Outlook.

Dim myOlApp As New Outlook.Application
Dim WithEvents mySync As Outlook.SyncObject
Dim myForm As New Form1

Sub Initialize_handler()
    Set mySync = myOlApp.Session.SyncObjects.Item(1)
End Sub

Private Sub mySync_SyncStart()
    Form1.Label1.Caption = "Synchronization started"
End Sub
ViewAdd Event

Occurs when a view is added to the collection. Microsoft Outlook creates the new view and passes it to this event.

Sub expression_ ViewAdd(ByVal View As View)

expression A variable which references an object of type Views declared with events in a class module.

View The new view added to the collection prior to this event.
Example

The following example displays the view’s name and saves it. Use the **Save** method after the properties have been modified to save the changes to the view.

```vba
Sub tblView_ViewAdd(ByVal View As View)
' Displays name of new view

    With View
        MsgBox .Name & " Was created."
        .Save
    End With

End Sub
```
ViewRemove Event

Occurs when a view has been removed from the specified collection.

Sub expression_ ViewRemove(ByVal View As View)

expression  A variable which references an object of type Views declared with events in a class module.

View  The view which was removed from the collection prior to this event.
Example

The following example displays the name of the view that has been removed from the collection.

Sub tblView_ViewRemove(ByVal View As View)
    'Displays view name
    With View
        MsgBox "The view: " & .Name & " has been removed."
    End With
End Sub
ViewSwitch Event

Occurs when the view in the explorer changes, either as a result of user action or through program code. This event is not available in VBScript.

Sub object_ViewSwitch()

object An expression that evaluates to an Explorer object.
Example

This example hides the preview pane if it is visible when the user switches to the Messages with AutoPreview view. The sample code must be placed in a class module, and the Initialize_handler routine must be called before the event procedure can be called by Microsoft Outlook.

Dim myolapp As New Outlook.Application
Dim WithEvents myOlExpl As Outlook.Explorer

Sub Initialize_handler()
    Set myOlExpl = myolapp.ActiveExplorer
End Sub

Private Sub myOlExpl_ViewSwitch()
    If myOlExpl.CurrentView = "Messages with AutoPreview" And myOlEx
        myOlExpl.ShowPane olPreview, False
    End If
End Sub
Write Event

Occurs when a Microsoft Outlook item is saved, either explicitly (for example, using the `Save` or `SaveAs` methods) or implicitly (for example, in response to a prompt when closing the item's inspector).

**Sub object_Write(Cancel As Boolean)**

*object*  An expression that evaluates to one of the objects in the Applies To list. In VBScript, use the word `Item`.

*Cancel*  Optional `Boolean` (not used in VBScript). `False` when the event occurs. If the event procedure sets this argument to `True`, the save operation is not completed.
Remarks

In VBScript, if you set the return value of this function to False, the save operation is not completed.
Example

This VBScript example uses the **Write** event and warns the user that the **item** is about to be saved and will overwrite any existing item and, depending on the user's response, either allows the operation to continue or stops it.

```vbnet
Function Item_Write()
    myMsg = "The item is about to be saved. Do you wish to overwrite"
    myResult = MsgBox(myMsg, 289, "Save")
    If myResult = 1 Then
        Item_Write = True
    Else
        Item_Write = False
    End If
End Function
```
Using events with Automation

To create an event handler for Microsoft Outlook objects in Microsoft Visual Basic or Microsoft Visual Basic for Applications in another application, you need to complete the following four steps:

1. Set a reference to the Microsoft Outlook Object Library.
2. Declare an object variable to respond to the events.
3. Write the specific event procedures.
4. Initialize the declared object.

Learn about working with events in Outlook Visual Basic for Applications.
Set the Reference to the Outlook Object Library

Before you can use an Outlook object in Visual Basic or Visual Basic for Applications code, you must first set a reference to the Outlook Object Model in the **References** dialog box. For more information about using this dialog box, see the online Help for your programming environment.
Declare the Object Variable

Once you’ve referenced the object model library, you must declare variables that reference the object you want to use. You can declare the variable in the module in which the object will be used (that is, the module containing the event-handler procedure), but more commonly you’ll declare it in a class module so it can be used in any module in your program.

For example, to declare an object variable for the Application object in a class module, you use code like the following.

Public WithEvents myOlApp As Outlook.Application

You must use the WithEvents keyword to specify that the object variable will be used to respond to events triggered by the object.
Write the Event Procedure

After the new object has been declared with events, it appears in the Object list in the class module Code window, and you can select the object’s event procedures from the Procedures/Events list. For example, when you select the ItemSend event for an Application object declared as myOlApp, the following empty procedure appears in the Code window.

Private Sub myOlApp_ItemSend(Item as Object, Cancel as Boolean)
End Sub
**Initialize the Declared Object**

Before the procedure will run, you must connect the declared object (in this example, myOlApp) with the **Application** object. If you declared the object in a class module named EventClassModule, then you can use the following code in any module.

```vbs
Dim myClass as New EventClassModule

Sub Register_Event_Handler()
    Set myClass.myOlApp = CreateObject("Outlook.Application")
End Sub
```

When the Register_Event_Handler procedure is run, the myOlApp object in the form or class module points to the Outlook **Application** object, and the event procedure will run when the event occurs.
The following code example provides the empty event procedures required to implement a COM add-in.

Implements IDTExtensibility2

Private Sub IDTExtensibility2_OnAddInsUpdate(custom() As Variant)
' Occurs when the set of connected COM add-ins changes, that is when any other add-in is connected or disconnected.
' The custom argument is ignored.
End Sub

Private Sub IDTExtensibility2_OnBeginShutdown(custom() As Variant)
' If the COM add-in is connected, occurs when Outlook begins its shutdown routines.
' The custom argument is ignored.
End Sub

Private Sub IDTExtensibility2_OnConnection(ByVal Application As Object, ByVal ConnectMode As AddInDesignerObjects.ext_ConnectMode, ByVal AddInInst As Object, custom() As Variant)
' Occurs when the COM add-in is connected.
' The Application argument is the Outlook Application object.
' The ConnectMode argument specifies how the COM add-in was connected.
' It can be
'   ext_cm_AfterStartup   Add-in was connected after Outlook started, or the Connect property of the corresponding COMAddIn object was set to True
'   ext_cm_Startup       Add-in was connected on startup
'   ext_cm_External
'   ext_cm_CommandLine
' The AddInInst argument is the COMAddIn object that refers to the current instance of the add-in itself.
' The custom argument is ignored.
End Sub

Private Sub IDTExtensibility2_OnDisconnection(ByVal RemoveMode As AddInDesignerObjects.ext_DisconnectMode, custom() As Variant)
' Occurs when the COM add-in is disconnected.
' The RemoveMode argument specifies how the COM add-in was disconnected.
' It can be
'   ext_dm_HostShutdown   Add-in was disconnected when Outlook
' closed.
' ext_dm_UserClosed  Add-in was disconnected when the user
cleared the corresponding check box in
COM Add-ins dialog box, or the Connect
property of the corresponding COMAddI
object was set to False.
' The custom argument is ignored.
End Sub

Private Sub IDTExtensibility2_OnStartupComplete(custom() As Variant)
' If the COM add-in connects at startup, occurs when Outlook complet
its startup routines. This event does not occur if the COM add-in
connected when Outlook loads, even when the user connects the add-
the COM Add-ins dialog box.
' The custom argument is ignored.
End Sub
Outlook Item Objects

Outlook items include

- AppointmentItem
- ContactItem
- DistListItem
- DocumentItem
- JournalItem
- MailItem
- MeetingItem
- NoteItem
- PostItem
- RemoteItem
- ReportItem
- TaskItem
- TaskRequestAcceptItem
- TaskRequestDeclineItem
- TaskRequestItem
- TaskRequestUpdateItem
GetNamespace Method

- Returns a NameSpace object of the specified type.

expression.GetNamespace(Type)

expression   Required. An expression that returns an Application object.

Type   Required String. The type of name space to return.
Remarks

The only supported name space type is "MAPI".
Example

This Visual Basic for Applications example uses `GetNameSpace` to obtain the MAPI `NameSpace` object.

```vba
Set myOlApp = CreateObject("Outlook.Application")
Set myNameSpace = myOlApp.GetNameSpace("MAPI")
```

If you use VBScript, you do not create the `Application` object. This example shows how to perform the same task using VBScript.

```vbs
Set myNameSpace = Application.GetNamespace("MAPI")
```
You can use an OLE programmatic identifier (sometimes called a ProgID) to create an Automation object. The following tables list OLE programmatic identifiers for ActiveX controls, Microsoft Office applications, and Microsoft Office Web Components.

**ActiveX Controls**

**Microsoft Access**

**Microsoft Excel**

**Microsoft Graph**

**Microsoft Office Web Components**

**Microsoft Outlook**

**Microsoft PowerPoint**

**Microsoft Word**
## ActiveX Controls

To create the ActiveX controls listed in the following table, use the corresponding OLE programmatic identifier.

<table>
<thead>
<tr>
<th>To create this control</th>
<th>Use this identifier</th>
</tr>
</thead>
<tbody>
<tr>
<td>CheckBox</td>
<td>Forms.CheckBox.1</td>
</tr>
<tr>
<td>ComboBox</td>
<td>Forms.ComboBox.1</td>
</tr>
<tr>
<td>CommandButton</td>
<td>Forms.CommandButton.1</td>
</tr>
<tr>
<td>Frame</td>
<td>Forms.Frame.1</td>
</tr>
<tr>
<td>Image</td>
<td>Forms.Image.1</td>
</tr>
<tr>
<td>Label</td>
<td>Forms.Label.1</td>
</tr>
<tr>
<td>ListBox</td>
<td>Forms.ListBox.1</td>
</tr>
<tr>
<td>MultiPage</td>
<td>Forms.MultiPage.1</td>
</tr>
<tr>
<td>OptionButton</td>
<td>Forms.OptionButton.1</td>
</tr>
<tr>
<td>ScrollBar</td>
<td>Forms.ScrollBar.1</td>
</tr>
<tr>
<td>SpinButton</td>
<td>Forms.SpinButton.1</td>
</tr>
<tr>
<td>TabStrip</td>
<td>Forms.TabStrip.1</td>
</tr>
<tr>
<td>TextBox</td>
<td>Forms.TextBox.1</td>
</tr>
<tr>
<td>ToggleButton</td>
<td>Forms.ToggleButton.1</td>
</tr>
</tbody>
</table>
**Microsoft Access**

To create the Microsoft Access objects listed in the following table, use one of the corresponding OLE programmatic identifiers. If you use an identifier without a version number suffix, you create an object in the most recent version of Access available on the machine where the macro is running.

<table>
<thead>
<tr>
<th>To create this object</th>
<th>Use one of these identifiers</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Application</strong></td>
<td>Access.Application</td>
</tr>
<tr>
<td><strong>CurrentData</strong></td>
<td>Access.CodeData, Access.CurrentData</td>
</tr>
<tr>
<td><strong>DefaultWebOptions</strong></td>
<td>Access.DefaultWebOptions</td>
</tr>
</tbody>
</table>
Microsoft Excel

To create the Microsoft Excel objects listed in the following table, use one of the corresponding OLE programmatic identifiers. If you use an identifier without a version number suffix, you create an object in the most recent version of Excel available on the machine where the macro is running.

<table>
<thead>
<tr>
<th>To create this object</th>
<th>Use one of these identifiers</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application</td>
<td>Excel.Application</td>
<td></td>
</tr>
<tr>
<td>Workbook</td>
<td>Excel.Addln</td>
<td></td>
</tr>
<tr>
<td>Workbook</td>
<td>Excel.Chart</td>
<td></td>
</tr>
<tr>
<td>Workbook</td>
<td>Excel.Sheet</td>
<td></td>
</tr>
</tbody>
</table>

- Returns a workbook containing two worksheets; one for the chart and one for its data. The chart worksheet is the active worksheet.
- Returns a workbook with one worksheet.
**Microsoft Graph**

To create the Microsoft Graph objects listed in the following table, use one of the corresponding OLE programmatic identifiers. If you use an identifier without a version number suffix, you create an object in the most recent version of Graph available on the machine where the macro is running.

<table>
<thead>
<tr>
<th>To create this object</th>
<th>Use one of these identifiers</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Application</strong></td>
<td>MSGraph.Application</td>
</tr>
<tr>
<td><strong>Chart</strong></td>
<td>MSGraph.Chart</td>
</tr>
</tbody>
</table>
Microsoft Office Web Components

To create the Microsoft Office Web Components objects listed in the following table, use one of the corresponding OLE programmatic identifiers. If you use an identifier without a version number suffix, you create an object in the most recent version of Microsoft Office Web Components available on the machine where the macro is running.

<table>
<thead>
<tr>
<th>To create this object</th>
<th>Use one of these identifiers</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChartSpace</td>
<td>OWC.Chart</td>
</tr>
<tr>
<td>DataSourceControl</td>
<td>OWC.DataSourceControl</td>
</tr>
<tr>
<td>ExpandControl</td>
<td>OWC.ExpandControl</td>
</tr>
<tr>
<td>PivotTable</td>
<td>OWC.PivotTable</td>
</tr>
<tr>
<td>RecordNavigationControl</td>
<td>OWC.RecordNavigationControl</td>
</tr>
<tr>
<td>Spreadsheet</td>
<td>OWC.Spreadsheet</td>
</tr>
</tbody>
</table>
Microsoft Outlook

To create the Microsoft Outlook object given in the following table, use one of the corresponding OLE programmatic identifiers. If you use an identifier without a version number suffix, you create an object in the most recent version of Outlook available on the machine where the macro is running.

<table>
<thead>
<tr>
<th>To create this object</th>
<th>Use one of these identifiers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application</td>
<td>Outlook.Application</td>
</tr>
</tbody>
</table>
Microsoft PowerPoint

To create the Microsoft PowerPoint object given in the following table, use one of the corresponding OLE programmatic identifiers. If you use an identifier without a version number suffix, you create an object in the most recent version of PowerPoint available on the machine where the macro is running.

<table>
<thead>
<tr>
<th>To create this object</th>
<th>Use one of these identifiers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application</td>
<td>PowerPoint.Application</td>
</tr>
</tbody>
</table>
**Microsoft Word**

To create the Microsoft Word objects listed in the following table, use one of the corresponding OLE programmatic identifiers. If you use an identifier without a version number suffix, you create an object in the most recent version of Word available on the machine where the macro is running.

<table>
<thead>
<tr>
<th>To create this object</th>
<th>Use one of these identifiers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application</td>
<td>Word.Application</td>
</tr>
<tr>
<td>Global</td>
<td>Word.Global</td>
</tr>
</tbody>
</table>