## **New Objects**

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

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

#### Object

#### Description

**Conflict** Represents an Outlook item that is in conflict with another Outlook item.

**Conflicts** Represents a collection of all Outlook items that are in conflict with a particular Outlook item.

# **New Properties (Alphabetical List)**

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

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

<b>New Property</b>	Object(s)	
<u>AutoResolvedWinner</u>	AppointmentItem, ContactItem, DistListItem, DocumentItem, JournalItem, MailItem, MeetingItem, NoteItem, PostItem, RemoteItem, ReportItem, TaskItem, TaskRequestAcceptItem, TaskRequestDeclineItem, TaskRequestItem, TaskRequestUpdateItem	
<u>Conflicts</u>	AppointmentItem, ContactItem, DistListItem, DocumentItem, JournalItem, MailItem, MeetingItem, NoteItem, PostItem, RemoteItem, ReportItem, TaskItem, TaskRequestAcceptItem, TaskRequestDeclineItem, TaskRequestItem, TaskRequestUpdateItem	
<b>EnableSharedAttachments</b>	MailItem	
<b>ExchangeConnectionMode</b>	NameSpace	
<b>FlagIcon</b>	MailItem, MeetingItem	
<b>HasCoverSheet</b>	MailItem	
<b>HasPicture</b>	ContactItem	
<u>IsIPFax</u>	MailItem	
<u>IsSharePointFolder</u>	MAPIFolder	
<b>MeetingWorkspaceURL</b>	AppointmentItem, MeetingItem	
Permission	MailItem	
SenderEmailAddress	MailItem, MeetingItem, PostItem	

SenderEmailType ShowItemCount MailItem, MeetingItem, PostItem MAPIFolder

# **New Properties (by Object)**

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

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

Object	<b>New Properties</b>	
	AutoResolvedWinner	
AppointmentItem	Conflicts, MeetingWorkspaceURL	
ContactItem	<u>AutoResolvedWinner, Conflicts, HasPicture</u>	
DistListItem	<u>AutoResolvedWinner, Conflicts</u>	
DocumentItem	<u>AutoResolvedWinner, Conflicts</u>	
JournalItem	<u>AutoResolvedWinner, Conflicts</u>	
MailItem	AutoResolvedWinner, Conflicts, EnableSharedAttachments, FlagIcon, HasCoverSheet, IsIPFax, Permission, SenderEmailAddress, SenderEmailType	
MAPIFolder	IsSharePointFolder, ShowItemCount	
MeetingItem	AutoResolvedWinner, Conflicts, FlagIcon, MeetingWorkspaceURL, SenderEmailAddress, SenderEmailType	
NameSpace	ExchangeConnectionMode	
NoteItem	AutoResolvedWinner, Conflicts	
PostItem	<u>AutoResolvedWinner, Conflicts,</u> <u>SenderEmailAddress, SenderEmailType</u>	
RemoteItem	AutoResolvedWinner, Conflicts	
ReportItem	AutoResolvedWinner, Conflicts	
TaskItem	AutoResolvedWinner, Conflicts	
TaskRequestAcceptItem	AutoResolvedWinner, Conflicts	
TaskRequestDeclineItem	AutoResolvedWinner, Conflicts	

TaskRequestItemAutoResolvedWinner, ConflictsTaskRequestUpdateItemAutoResolvedWinner, Conflicts

# **New Methods (Alphabetical List)**

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

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

<b>New Method</b>	Object
AddPicture	ContactItem
AddStoreEx	NameSpace
<b>DeselectFolder</b>	Explorer
<b>IsFolderSelected</b>	Explorer
<b><u>RemovePicture</u></b>	ContactItem
<u>SelectFolder</u>	Explorer

# **New Methods (by Object)**

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

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

New MethodObjectContactItemAddPicture, RemovePictureExplorerDeselectFolder, IsFolderSelected, SelectFolderNameSpaceAddStoreEx

## **New Events**

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

The following table lists events added to the Office Outlook 2003 object model.

New Event Object <u>NewMailEx</u> Application

# Using Visual Basic for Applications in Outlook

Visual Basic for Applications (VBA) 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 Microsoft Visual Basic Scripting Edition (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 <u>Outlook forms</u>).

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 <u>writing an Outlook macro</u>, <u>responding to Outlook events</u>, and <u>designing Visual Basic for Applications forms</u>.

# 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 <u>objects</u>. The <u>Application</u> 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 OutlookBarShort
    If Shortcut.Name = "Notes" Then
    MsgBox "You cannot open the Notes folder."
    Cancel = True
```

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.

```
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 (VBA) program. Unlike an <u>Outlook form</u>, 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.

```
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(olFc
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 <u>Outlook object model</u> 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 <u>Application</u> 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 (VBA) 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 <u>Application</u> 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.

<u>Automating Outlook from a Visual Basic Application</u> 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 (VBA), and Microsoft Visual Basic Scripting Edition (VBScript) programs that a significant change has occurred. For example, Outlook events can notify a program when an item has been opened or when a new mail arrives in the InBox.

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 <u>Shortcuts pane</u>), 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.

```
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 <u>writing an application-level event handler</u>.

### **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 <u>Outlook COM Add-in Template</u>.

### **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\SampleAd
"FriendlyName"="Sample Add-in"
"Description"="Sample Outlook Add-In"
"LoadBehavior"=dword:0000008
```

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

#### Value

#### Description

- 2 Load at startup. The COM add-in is to be loaded and connected when Outlook starts.
- Load on demand. The COM add-in is to be loaded and connected only when the user requests it, such as by using the COM Add-ins dialog box.
   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
- 16 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 example).

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**

- 0 Disconnected
- 1 Connected

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.

# Security notes for COM add-in developers

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

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

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

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

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

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

You can avoid the display of security warnings by deriving all objects, properties, and methods from the **Application** object passed to the

**OnConnection** procedure of the add-in. Outlook 2003 trusts only the **Application** object passed to the **OnConnection** procedure of the add-in. If you create a new **Application** object, for example, by using the **CreateObject** method, that object and any of its subordinate objects, properties, and methods will not be trusted and the blocked properties and methods will raise security warnings.

#### **New Object Model Blocks**

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

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

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

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

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



# **Actions Object**

Multiple objects  $\[ \] \underline{Actions} \]$ 

A collection of <u>Action</u> objects that represent all the specialized actions that can be executed on an Outlook <u>item</u>.

#### **Using the Actions Object**

Use the <u>Actions</u> 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 (VBA) 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
```

# **AddressEntries Object**

Multiple objects <sup>L</sup><u>AddressEntries</u> <sup>L</sup><u>AddressEntry</u> <sup>L</sup>Multiple objects

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)
```

### Remarks

If a program tries to reference any type of recipient information by using the Outlook object model, a dialog box is displayed that asks you to confirm access to this information. You can allow access to the Address Book or recipient information for up to ten minutes after you receive the dialog box. This allows features, such as mobile device synchronization, to be completed.

You receive the confirmation dialog box when a solution tries to programmatically access the **AddressEntries** object.



# **AddressLists Object**

NameSpace <sup>L</sup>AddressLists <sup>L</sup>AddressList <sup>L</sup>AddressEntries

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 <u>session</u>.

### Using the AddressLists Object

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

Set myAddressLists = myNameSpace.AddressLists



# **Attachments Object**

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

An object containing <u>Attachment</u> objects that represent the attachments in an Outlook <u>item</u>.

#### **Using the Attachments Object**

Use the <u>Attachments</u> property to return the **Attachments** collection for any Outlook item (except notes).

Use the **<u>Add</u>** 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 (VBA) example creates a new mail message, attaches a Q496.xls as an attachment (not a link), and gives the attachment a descriptive caption.

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

# **Conflicts Object**

Multiple objects  $\[ \] Conflicts \]$ 

The **Conflicts** object is a collection of **Conflict** objects that represent all Microsoft Outlook items that are in conflict with a particular Outlook item.

### **Using the Conflicts Object**

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

Use the **Count** property of the **Conflicts** object to determine if the item is invloved in a conflict. A non-zero value indicates conflict.

Use the **Item** method to retrieve a particular conflict item from the **Conflicts** collection object.

Use the **GetFirst**, **GetNext**, **GetPrevious**, and **GetLast** methods to traverse the Conflicts collection.

The following Microsoft Visual Basic for Applications (VBA) example uses the **Count** property of the **Conflicts** object to determine if the item is involved in any conflict. To run this example, make sure an e-mail item is open in the active window.

# **Exceptions Object**

RecurrencePattern Exceptions

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

### **Explorers Object**

Explorers <sup>L</sup>Explorer <sup>L</sup>Multiple objects

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 and Microsoft Visual Basic for Applications (VBA).

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

The following example shows how to retrieve the **Explorers** object in Microsoft Visual Basic Scripting Edition (VBScript).

Set myExplorers= Application.Explorers

### **Folders Object**

<u>Folders</u> <u>MAPIFolder</u> <u>View</u>

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 (VBA) example returns the folder named Old Contacts.

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

Set myNewFolder = myFolder.Folders(1)

### **Inspectors Object**

Inspectors LInspector

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 Microsoft Visual Basic Scripting Edition (VBScript).

Set myInspectors= Application.Inspectors

# **ItemProperties Collection**

Multiple objects LitemProperties

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.

```
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.

# **Links Object**

Multiple objects Links

The **Links** object contains a set of **Link** objects representing all <u>items</u> 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



### **OutlookBarGroups Object**

OutlookBarGroups <sup>L</sup>OutlookBarGroup

The **OutlookBarGroups** object contains a set of <u>**OutlookBarGroup**</u> objects representing all groups in the <u>Shortcuts pane</u>.

### Using the OutlookBarGroups Object

Use the <u>**Groups</u>** property to return the **OutlookBarGroups** object from the <u>**OutlookBarStorage**</u> object. For example:</u>

Set myGroups = myOutlookBarStorage.Groups



# **OutlookBarShortcuts Object**

OutlookBarShortcuts <sup>L</sup>OutlookBarShortcut

The **OutlookBarShortcuts** <u>collection</u> contains a set of <u>**OutlookBarShortcut**</u> objects representing all <u>shortcuts</u> in a group in the <u>Shortcuts pane</u>.

### Using the OutlookBarShortcuts Object

Use the **<u>Shortcuts</u>** property to return the **OutlookBarShortcuts** collection object from the **<u>OutlookBarGroup</u>** object. For example:

Set myShortcuts = myOutlookBarGroup.Shortcuts

# **Pages Object**

#### **Pages**

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.

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").

Set myPage = myPages.Add

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

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

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

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

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

```
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.



### **Panes Object**

Explorer Panes

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 <u>Item</u> method to retrieve a specific pane. To retrieve the <u>OutlookBarPane</u> object representing the <u>Shortcuts pane</u>, use the following:

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

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

# **PropertyPages Object**

<u>PropertyPages</u> <u>PropertyPage</u>

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.

# **Recipients Object**

Multiple objects <sup>L</sup><u>Recipients</u> <u>Recipient</u> <u>AddressEntry</u>

The **Recipients** object contains **<u>Recipient</u>** objects.

#### Using the Recipients Object

Use the **<u>Recipients</u>** property to return the **Recipients** object of an **<u>AppointmentItem</u>**, **<u>JournalItem</u>**, **<u>MailItem</u>**, **<u>MeetingItem</u>** or <u>**TaskItem**</u> 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").

```
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.

```
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.

#### Remarks

Outlook blocks code that attempts to access the **Recipients** object for security reasons. If you run a third-party add-in, custom solution, or other program that uses the **Recipients** object in Office Outlook 2003, you may receive the following warning:

A program is trying to access e-mail addresses you have stored in Outlook. Do you want to allow this? If this is unexpected, it may be a virus and you should choose "No".

### **Reminders Collection**

<u>Reminders</u> <u>Reminder</u>

A collection of all the **<u>Reminder</u>** objects in a Microsoft Outlook application that represents the reminders for all pending 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.

```
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
                strReport = strReport & objRem.Caption & vbCr
            End If
        Next objRem
        'Display report in dialog
        MsgBox strTitle & vbCr & vbCr & 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 appointment item and sets the ReminderSet property to **True**, adding a new **Reminder** object to the **Reminders** collection.

```
Sub AddAppt()
'Adds a new appointment and reminder to the reminders collection
Dim olApp As Outlook.Application
Dim objApt As AppointmentItem
Set olApp = Outlook.Application
Set objApt = olApp.CreateItem(olAppointmentItem)
objApt.ReminderSet = True
objApt.Subject = "Tuesday's meeting"
objApt.Save
End Sub
```

### **Results Collection**

#### <u>Results</u>

Stores data and results returned by the <u>Search</u> object and the <u>AdvancedSearch</u> method. The **Results** object contains properties and methods that allow you to view and manipulate data. For example the <u>GetNext</u>, <u>GetPrevious</u>, <u>GetFirst</u>, and <u>GetLast</u> methods allow you to search through the results and view the data by field. The <u>Sort</u> 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.

Private Sub Application\_AdvancedSearchComplete(ByVal SearchObject As

End Sub

# SyncObjects Object

NameSpace LSyncObjects

The **SyncObjects** object contains a set of **<u>SyncObject</u>** objects representing the Send\Receive groups 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. However, note that you can add one Send/Receive group using the **AppFolders** property which will create a Send/Receive group called **Application Folders**.



# **UserProperties Object**

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

An object containing **<u>UserProperty</u>** objects that represent the custom properties of an Outlook <u>item</u>.

#### **Using The UserProperties Object**

Use the <u>UserProperties</u> 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.

```
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.

### **Views Collection**

<u>Views</u> <u>View</u>

A collection of all <u>View</u> objects in the current folder.

#### Using the Views collection

Use the **Views** property of the **MAPIFolder** object 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. Before running this example, make sure a view by the name 'Table View' exists.

```
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 **ollconView** 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", __
```

End Sub

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.

```
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



# **Action Object**

### Actions <sup>L</sup>Action

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

#### **Using the Action Object**

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

The following Visual Basic for Applications (VBA) 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
```

# **AddressEntry Object**

Multiple objects <sup>L</sup><u>AddressEntry</u> <sup>L</sup>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)
```

#### Remarks

If a program tries to reference any type of recipient information by using the Outlook object model, a dialog box is displayed that asks you to confirm access to this information. You can allow access to the Address Book or recipient information for up to ten minutes after you receive the dialog box. This allows features, such as mobile device synchronization, to be completed.

You receive the confirmation dialog box when a solution tries to programmatically access the **AddressEntry** object.



# AddressList Object

<u>AddressLists</u> <u>AddressList</u> <u>AddressEntries</u>

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 <u>session</u>. The entire hierarchy is available through the parent <u>AddressLists</u> object.



# **Application Object**

#### **Application**

Represents the entire Microsoft Outlook application. This is the only object in the hierarchy that can be returned by using the **<u>CreateObject</u>** 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 <u>item</u> created by using <u>CreateItem</u>, 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.

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

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

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

# **AppointmentItem Object**

Exception <sup>L</sup><u>AppointmentItem</u> <sup>L</sup>Multiple objects

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 **<u>CreateItem</u>** method to create an **AppointmentItem** object that represents a new appointment.

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

```
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.

#### Remarks

If a program tries to reference any type of recipient information by using the Outlook object model, a dialog box is displayed that asks you to confirm access to this information. You can allow access to the Address Book or recipient information for up to ten minutes after you receive the dialog box. This allows features, such as mobile device synchronization, to be completed.

You receive the confirmation dialog box when a solution tries to programmatically access the following properties of the **AppointmentItem** object:

- Organizer
- RequiredAttendees
- OptionalAttendees
- Resources
- NetMeetingOrganizerAlias



## **Attachment Object**

Attachments LAttachment

Represents a document or link to a document contained in an Outlook item.

#### **Using the Attachment Object**

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

Use the **<u>Add</u>** method to add an attachment to an item.

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

### **Conflict Object**

### Conflicts Conflict

Represents a Microsoft Outlook item that is in conflict with another Outlook item. Each Outlook item has a **Conflicts** collection object associated with it that represents all the items that are in conflict with that item.

#### **Using the Conflict Object**

Use the **Item** method to retrieve a particular **Conflict** object from the **Conflicts** collection object, for example:

```
Set myConflictItem = myConflicts.Item(1)
```

### **ContactItem Object**

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

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 <u>CreateItem</u> method to create a **ContactItem** object that represents a new contact.

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

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

The following Microsoft Visual Basic Scripting Edition (VBScript) example returns a new contact.

```
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.

#### Remarks

If a program tries to reference any type of recipient information by using the Outlook object model, a dialog box is displayed that asks you to confirm access to this information. You can allow access to the Address Book or recipient information for up to ten minutes after you receive the dialog box. This allows features, such as mobile device synchronization, to be completed.

You receive the confirmation dialog box when a solution tries to programmatically access the following properties of the **ContactItem** object:

- Email1Address
- Email1AddressType
- Email1DisplayName
- Email1EntryID
- Email2Address
- Email2AddressType
- Email2DisplayName
- Email2EntryID
- Email3Address
- Email3AddressType
- Email3DisplayName
- Email3EntryID
- NetMeetingAlias
- ReferredBy

# **DistListItem Object**

DistListItem Multiple objects

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 <u>CreateItem</u> method to create a **DistListItem** object that represents a new distribution list. The following Microsoft Visual Basic for Applications (VBA) example creates and displays a new distribution list.

```
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.

```
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
```



## **DocumentItem Object**

DocumentItem <sup>L</sup>Multiple objects

A **DocumentItem** object is any document other than a Microsoft Outlook <u>item</u> 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.

**Note** When you try to programmatically add a user-defined property to a **DocumentItem** object, you receive the following error message: "Property is read-only." This is because the Outlook object model does not support this functionality.

#### Example

The following Visual Basic for Applications (VBA) example shows how to create a DocumentItem.

```
Sub AddDocItem()
    Dim outApp As New Outlook.Application
    Dim nsp As Outlook.NameSpace
    Dim mpfInbox As Outlook.MAPIFolder
    Dim doci As Outlook.DocumentItem
    Set nsp = outApp.GetNamespace("MAPI")
    Set mpfInbox = nsp.GetDefaultFolder(olFolderInbox)
    Set doci = mpfInbox.Items.Add(olWordDocumentItem)
    doci.Subject = "Word Document Item"
    doci.Save
End Sub
```

## **Exception Object**

Exceptions Lexception

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 (VBA):

```
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 Microsoft Visual Basic Scripting Edition (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)
```

## **Explorer Object**

Explorers Lexplorer

Represents 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 <u>ActiveExplorer</u> method to return the object representing the currently active explorer (if there is one).
- Use the <u>GetExplorer</u> 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.



# **FormDescription Object**

Multiple objects <sup>L</sup>FormDescription

Contains the general properties of a Microsoft Outlook <u>form</u>. 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 <u>item</u>, 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**

Inspectors LInspector

Represents the window in which an Outlook <u>item</u> is displayed.

#### **Using the Inspector Object**

- Use the <u>ActiveInspector</u> method to return the object representing the currently active inspector (if there is one).
- Use the <u>GetInspector</u> property to return the **Inspector** object associated with an item.
- Use the **Display** method to display an item in its associated inspector.

# **ItemProperty Object**

ItemProperties LItemProperty

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**

<u>Items</u>

An object containing <u>Microsoft Outlook item objects</u> 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.

**Note** In Office Outlook 2003, the items in the **Items** collection object are not guaranteed to be in any particular order.

The following Microsoft Visual Basic for Applications (VBA) example returns the first item in the **Inbox** with the Subject "Need your advice."

The following VBA example returns the first item in the **Inbox**. In Office Outlook 2003, the **Items** object returns the items in an Offline Folders file (.ost) in the reverse order.

The following Microsoft Visual Basic Scripting Edition (VBScript) example returns the first item in the **Inbox**. In Office Outlook 2003, the **Items** object returns the items in an Offline Folders file (.ost) in the reverse order.

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

# JournalItem Object

JournalItem <sup>L</sup>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 **<u>CreateItem</u>** 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

### Links Link

Represents an <u>item</u> that is linked to another Microsoft Outlook item. Each item has a <u>Links</u> 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")

# **MailItem Object**

MailItem <sup>L</sup>Multiple objects

Represents a mail message in an Inbox (mail) folder.

#### **Using the MailItem Object**

Use the <u>CreateItem</u> method to create a **MailItem** object that represents a new mail message. The following example creates and displays a new mail message.

```
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.

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

#### Remarks

If a program tries to reference any type of recipient information by using the Outlook object model, a dialog box is displayed that asks you to confirm access to this information. You can allow access to the Address Book or recipient information for up to ten minutes after you receive the dialog box. This allows features, such as mobile device synchronization, to be completed.

You receive the confirmation dialog box when a solution tries to programmatically access the following properties of the **MaiItem** object:

- SentOnBehalfOfName
- SenderName
- ReceivedByName
- ReceivedOnBehalfOfName
- ReplyRecipientNames
- To
- CC
- BCC
- Body
- HTMLBody
- Recipients
- SenderEmailAddress



# **MAPIFolder Object**

Multiple objects <sup>L</sup><u>MAPIFolder</u> <u>L</u><u>View</u>

Represents a Microsoft Outlook folder. A **MAPIFolder** object can contain other **MAPIFolder** objects, as well as Outlook <u>items</u>. 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.

**Note** Search folders are not **MAPIFolder** objects and therefore, the methods and properties of a **MAPIFolder** object will not work on search folders.

### **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 <u>GetDefaultFolder</u> (*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, olFolderTasks, olPublicFoldersAllPublicFolders, and olFolderJunk.

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 <u>AppointmentItem</u> objects and the Contacts folder will only contain <u>ContactItem</u> and <u>DistListItem</u> 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**

MeetingItem <sup>L</sup>Multiple objects

Represents an <u>item</u> 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 <u>MeetingStatus</u> property of an <u>AppointmentItem</u> 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 <u>CreateItem</u> 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**

NameSpace <sup>L</sup>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 <u>GetNameSpace</u> ("MAPI") to return the Outlook NameSpace object from the <u>Application</u> object.

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

### **NoteItem Object**

#### NoteItem <sup>L</sup>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 **<u>CreateItem</u>** method to create a **NoteItem** object that represents a new note. The following Microsoft Visual Basic example returns a new note.

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

The following example shows how to create a **NoteItem** object using Microsoft Visual Basic Scripting Edition (VBScript).

```
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**

OutlookBarGroups OutlookBarGroup

Represents a group of <u>shortcuts</u> in the <u>Shortcuts pane</u> of an explorer window.

#### Using the OutlookBarGroup Object

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

Set myOlBarGroup = myOutlookBarGroups.Item("Other Shortcuts")



## **OutlookBarPane Object**

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

Represents the <u>Shortcuts pane</u> in an explorer window.

#### 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 **OutlookBarPane** object by name. For example:

Set myOlBarPane = myPanes.Item("OutlookBar")



## **OutlookBarShortcut Object**

OutlookBarShortcuts OutlookBarShortcut

Represents a <u>shortcut</u> in a group in the <u>Shortcuts pane</u>.

#### 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")



## **OutlookBarStorage Object**

OutlookBarPane \_\_OutlookBarStorage

Represents the storage for objects in the <u>Shortcuts pane</u>.

#### Using the OutlookBarStorage Object

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

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

Use the **<u>Groups</u>** property to retrieve the **<u>OutlookBarGroups</u>** object for the Shortcuts pane.

### **PostItem Object**

PostItem <sup>L</sup>Multiple objects

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

#### Using the PostItem Object

Use the **<u>CreateItem</u>** or **<u>CreateItemFromTemplate</u>** 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 L 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.

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.

```
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.

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

## **PropertyPageSite Object**

**PropertyPageSite** 

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.

Private myPropertyPageSite As Outlook.PropertyPageSite

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

Set myPropertyPageSite = Parent

Use the **OnStatusChange** method to notify Microsoft Outlook that the property page has changed.

# **Recipient Object**

Multiple objects <sup>L</sup><u>Recipients</u> <sup>L</sup><u>Recipient</u> <sup>L</sup><u>AddressEntry</u>

Represents a user or resource in Outlook, generally a mail message addressee.

#### **Using the Recipient Object**

Use **<u>Recipients</u>** (*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 (VBA) 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
```

#### Remarks

If a program tries to reference any type of recipient information by using the Outlook object model, a dialog box is displayed that asks you to confirm access to this information. You can allow access to the Address Book or recipient information for up to ten minutes after you receive the dialog box. This allows features, such as mobile device synchronization, to be completed.

You receive the confirmation dialog box when a solution tries to programmatically access the **Recipient** object.



### **RecurrencePattern Object**

RecurrencePattern Exceptions

Represents the pattern of incidence of recurring appointments and tasks for the associated <u>AppointmentItem</u> and <u>TaskItem</u> object.

#### **Using the RecurrencePattern Object**

Use the **<u>GetRecurrencePattern</u>** method to return the **RecurrencePattern** object associated with an **AppointmentItem** or **TaskItem** object.

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

The type of recurrence pattern is indicated by the **<u>RecurrenceType</u>** 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**.

RecurrenceType	Properties	Example
olRecursDaily	Interval	Every N days
	DayOfWeekMask	Every Tuesday, Wednesday, and Thursday
olRecursMonthly	Interval	Every N months
	DayOfMonth	The Nth day of the month
olRecursMonthNth	Interval	Every N months
	Instance	The Nth Tuesday
	DayOfWeekMask	Every Tuesday and Wednesday
olRecursWeekly	Interval	Every N weeks
	DayOfWeekMask	Every Tuesday, Wednesday, and Thursday
olRecursYearly	DayOfMonth	The Nth day of the month
	MonthOfYear	February
olRecursYearNth	Instance	The Nth Tuesday

DayOfWeekMaskTuesday, Wednesday, ThursdayMonthOfYearFebruary

## **Reminder Object**

Reminders LReminder

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 **<u>Reminders</u>** (*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
    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 appointment item and sets the **ReminderSet** property to **True**, adding a new **Reminder** object to the **Reminders** collection.

```
Sub AddAppt()
'Adds a new appointment and reminder to the reminders collection
Dim olApp As Outlook.Application
Dim objApt As AppointmentItem
Set olApp = Outlook.Application
Set objApt = olApp.CreateItem(olAppointmentItem)
objApt.ReminderSet = True
```

```
objApt.Subject = "Tuesday's meeting"
objApt.Save
```

End Sub

Use the **Reminders** collection's **<u>Remove</u>** 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**.



### **RemoteItem Object**

RemoteItem <sup>L</sup>Multiple objects

Represents a remote item in an **Inbox** (mail) folder. The **RemoteItem** object is similar to the <u>MailItem</u> 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. However, the headers in items contained in an Offline Folders file (.ost) cannot be accessed using the **RemoteItem** object.

#### 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 preexisting **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 trying 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**

<u>ReportItem</u> <sup>L</sup>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.

# **Search Object**

#### <u>Search</u>

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 **<u>Application</u>** object's **<u>AdvancedSearch</u>** method to return a **Search** object.

#### Example

The following Microsoft Visual Basic for Applications (VBA) 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 VBA 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**

Explorer LSelection

The **Selection** object contains a set of Microsoft Outlook items representing the <u>items</u> currently selected in an explorer.

### **Using the Selection Object**

Use the <u>Selection</u> property to return the **Selection** collection from the <u>Explorer</u> object. For example:

Set mySelectedItems = myExplorer.Selection

# SyncObject Object

SyncObjects SyncObject

Represents a Send\Receive group for a user. A Send\Receive group 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. However, note that you can add one Send/Receive group using the **AppFolders** property which will create a Send/Receive group called **Application Folders**.

## **TaskItem Object**

TaskItem 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 **<u>CreateItem</u>** method to create a **TaskItem** object that represents a new task.

The following Visual Basic for Applications (VBA) 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 Microsoft Visual Basic Scripting Edition (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.

### Remarks

If a program tries to reference any type of recipient information by using the Outlook object model, a dialog box is displayed that asks you to confirm access to this information. You can allow access to the Address Book or recipient information for up to ten minutes after you receive the dialog box. This allows features, such as mobile device synchronization, to be completed.

You receive the confirmation dialog box when a solution tries to programmatically access the following properties of the **TaskItem** object:

- ContactNames
- Delegator
- Owner
- StatusUpdateRecipients
- StatusOnCompletionRecipients

## TaskRequestAcceptItem Object

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

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 <sup>L</sup>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 <sup>L</sup>Multiple objects

Represents an <u>item</u> 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 <u>Assign</u> and <u>Send</u> methods to a <u>TaskItem</u> 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 (VBA) 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**.

```
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 Microsoft Visual Basic Scripting Edition (VBScript).

```
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 **<u>GetAssociatedTask</u>** method to return the **TaskItem** object, and work directly with the **TaskItem** object to respond to the request.

## TaskRequestUpdateItem Object

TaskRequestUpdateItem <sup>L</sup>Multiple objects

Represents an item in an Inbox (mail) folder.

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



# **UserProperty Object**

<u>UserProperties</u> <u>UserProperty</u>

Represents a custom property of a Microsoft Outlook <u>item</u>.

### **Using The UserProperty Object**

Use <u>UserProperties</u> (*index*), where *index* is a name or index number, to return a single **UserProperty** object.

Use the <u>Add</u> method to create a new **UserProperty** for an item and add it to the <u>UserProperties</u> 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)
```

**Note** 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

#### Views

<sup>L</sup>View

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 (**ollconView**) 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. Before running this example, make sure a view by the name 'Table View' exists.

```
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.

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.

```
Public WithEvents myOlApp As Outlook.Application
Public WithEvents myOlExp As Outlook.Explorer
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

*expression* Required. An expression that returns an <u>Application</u> object.

### **Example**

The following Microsoft Visual Basic/Visual Basic for Applications (VBA) example uses the **Count** property and **Item** method of the **Selection** collection returned by the **Selection** property to display the senders of all mail items selected in the active explorer window. To run this example, you need to have at least one mail item selected in the active Explorer window.

Note You might receive an error if you select items other than a mail item such as task request as the **SenderName** property does not exist for a TaskRequestItem object.

```
Sub GetSelectedItems()
        Dim myOlApp As New Outlook.Application
        Dim myOlExp As Outlook.Explorer
        Dim myOlSel As Outlook.Selection
        Dim MsgTxt As String
        Dim x As Integer
        MsqTxt = "You have selected items from: "
        Set myOlExp = myOlApp.ActiveExplorer
        Set myOlSel = myOlExp.Selection
        For x = 1 To myOlSel.Count
                MsgTxt = MsgTxt & myOlSel.Item(x).SenderName & ";"
        Next x
        MsgBox MsgTxt
```

End Sub

# **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 <u>Application</u> object

#### Remarks

If you are using Microsoft Visual Basic Scripting Edition (VBScript) in a Microsoft Outlook form, you should typically use the GetInspector method to refer to the Inspector object associated with the form, for example:

Set myInspector = Item.GetInspector

#### Example

This Visual Basic for Applications (VBA) example uses the <u>ActiveInspector</u> method to obtain the currently active <u>Inspector</u> object and enables the display of keys in ToolTips in the inspector.

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 <u>Application</u> object.

## Example

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

```
Sub MinimizeActiveWindow()
        Dim myOlApp As New Outlook.Application
        If TypeName(myOlApp.ActiveWindow) = "Inspector" Then
            myOlApp.ActiveWindow.WindowState = olMinimized
        End If
End Sub
```



# Add Method

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

Creates a new action in the <u>Actions</u> collection and returns the new action as an <u>Action</u> object.

expression.Add

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

expression.Add(Type, Name, Address)

*expression* 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 attachment in the <u>Attachments</u> collection and returns the new attachment as an <u>Attachment</u> object.

expression.Add(Source, Type, Position, DisplayName)

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

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

*Type* Optional **String**. The type of the attachment.

**Position** Optional **String**. In e-mail messages using Microsoft Outlook Rich Text format, position where the attachment should be placed. A value of 1 for the **Position** parameter specifies that the attachment should be positioned at the beginning of the message body. A value 'n' greater than the number of characters in the body of the e-mail item specifies that the attachment should be placed at the end. A value of 0 makes the attachment hidden.

*DisplayName* Optional **String**. Display name for the attachment.

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)

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

*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 <u>OlFolderDisplayMode</u> constants:

OlFolderDisplayMode can be one of these OlFolderDisplayMode constants. olFolderDisplayFolderOnly olFolderDisplayNoNavigation olFolderDisplayNormal

**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, olPublicFoldersAllPublicFolders, or olFolderTasks. (The constants olFolderDeletedItems, olFolderOutbox, olFolderJunk, olFolderConflicts, olFolderLocalFailures, olFolderServerFailures, olFolderSyncIssues, 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 <u>Inspectors</u> <u>collection</u> 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 <u>ItemProperties</u> object.

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

*Type* Required <u>OlUserPropertyType</u>. The type of the new **ItemProperty**.

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

*AddToFolderFields* Optional **Variant**. Determines if the item property will be added to the folder fields.

*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: olWordDocumentItem, olExcelWorkSheetItem, olPowerPointShowItem, or any valid message class. Specify a MessageClass to create custom forms.

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

Links a contact item to another item by adding a <u>Link</u> object to the <u>Links</u> 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 <u>Shortcuts pane</u> and returns the new group as an <u>**OutlookBarGroup**</u> object.

### expression.Add(Name, Index)

*oexpression* Required. An expression that returns an <u>OutlookBarGroups</u> 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 Shortcuts pane. 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 Shortcuts pane and returns the new shortcut as an **OutlookBarShortcut** object.

### expression.Add(Target, Name, Index)

*expression* Required. An expression that returns an <u>**OutlookBarShortcuts</u>** collection object.</u>

*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 Shortcuts pane 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 **<u>PropertyPages</u>** 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 **<u>Recipients</u>** collection and returns the new recipient as a **<u>Recipient</u>** 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 <u>UserProperties</u> collection, and returns the new property as a <u>UserProperty</u> 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 <u>OlUserPropertyType</u>. 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.

Note You can only add user-defined fields to Outlook items. User-defined fields

cannot be added to Microsoft Office document items such as Microsoft Word, Microsoft Excel, or Microsoft PowerPoint files even though you can programmatically create those items using the **olOfficeDocItemsType** constants. You will receive an error when you try to programmatically add a user-defined field to a **DocumentItem** object.

Add method as it applies to the Views object.

Creates a new view in the **Views** collection, and returns the new view as a **View** object.**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 <u>OlViewType</u>. 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 **Attachments** object.

For e-mail messages in Microsoft Outlook Rich Text format, a value of 1 for the **Position** parameter specifies that the attachment should be positioned at the beginning of the message body. A value 'n' greater than the number of characters in the body of the mail item specifies that the attachment should be placed at the end. A value of 0 makes the attachment hidden.

The following Microsoft Visual Basic /Visual Basic for Applications (VBA) example creates a mail item, adds an attachment, and displays it. To run this example, make sure a file called Test.Doc exists in the C:\ folder.

```
Sub AddAttachment()
    Dim myOlApp As New Outlook.Application
    Dim myItem As Outlook.MailItem
    Dim myAttachments As Outlook.Attachments
    Set myItem = myOlApp.CreateItem(olMailItem)
    Set myAttachments = myItem.Attachments
    myAttachments.Add "C:\Test.doc", _
        olByValue, 1, "Test"
    myItem.Display
End Sub
```

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

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

The following Visual Basic/VBA example displays the **Drafts** folder in an explorer window without a <u>Navigation Pane</u> or <u>Folder List</u>.

```
Sub DisplayDrafts()
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
End Sub
```

If you use Microsoft Visual Basic Scripting Edition (VBScript) in an Outlook form, you do not create the <u>Application</u> object, and you cannot use named constants. This example shows how to perform the same task using VBScript code.

```
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.

This method is essentially identical to the **<u>GetInspector</u>** property.

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

```
Sub DisplayMyContacts()
Dim myOlApp As New Outlook.Application
Dim myFolder As MAPIFolder
Dim myItems As Items
Dim myRestrictItems As Items
Dim answer As String
Dim filter As String
Dim myInspector As Inspector
Dim x As Integer
 answer = InputBox("Enter the company name")
Set myFolder = myOlApp.GetNamespace("MAPI")
    .GetDefaultFolder(olFolderContacts)
 filter = "[MessageClass] = 'IPM.Contact' AND [CompanyName] = '" & a
Set myItems = myFolder.Items
Set myRestrictItems = myItems.Restrict(filter)
For x = 1 To myRestrictItems.Count
    Set myInspector = myOlApp.Inspectors.Add(myRestrictItems.Item(x)
   myInspector.Display
Next x
End Sub
```

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

This VBA example creates a new mail message and uses the **Add** method to add an <u>Action</u> to it. To run this example without any errors, replace 'Dan Wilson' with a valid recipient name.

```
Sub AddAction()
Dim myOlApp As New Outlook.Application
Dim myItem As Outlook.MailItem
Dim myAction As Outlook.Action
Set myItem = myOlApp.CreateItem(olMailItem)
Set myAction = myItem.Actions.Add
myAction.Name = "Link Original"
```

```
myAction.ShowOn = olMenuAndToolbar
myAction.ReplyStyle = olLinkOriginalItem
myItem.To = "Dan Wilson"
myItem.Send
End Sub
```

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

```
Set myItem = Application.CreateItem(0)
Set myAction = myItem.Actions.Add
myAction.Name = "Link Original"
myAction.ShowOn = 2
myAction.ReplyStyle = 4
myItem.To = "Kim Buhler"
myItem.Send
```

As it applies to the **Links** collection.

This Visual Basic/VBA 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.

```
Sub AddLink()
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
Dim myItems As Outlook. Items
Dim tempstr As String
 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 t
 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
End If
End Sub
```

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

```
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 ta
If tempstr <> "" Then
    tempstr = "[Full Name] = """ & tempstr & """"
    Set myItems = myFolder.Items.Restrict("[MessageClass] = 'IPM.Con
    Set myContact = myItems.Find(tempstr)
    myTask.Links.Add myContact
```

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

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

```
Sub AddContactsFolder()
Dim myOlApp As New Outlook.Application
Dim myNameSpace As Outlook.NameSpace
Dim myFolder As Outlook.MAPIFolder
Dim myNewFolder As Outlook.MAPIFolder
Set myOlApp = CreateObject("Outlook.Application")
Set myNameSpace = myOlApp.GetNamespace("MAPI")
Set myFolder = myNameSpace.GetDefaultFolder(olFolderContacts)
Set myNewFolder = myFolder.Folders.Add("My Contacts")
End Sub
```

If you use VBScript in an Outlook form, 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 VBA example uses the **Add** method to add three new folders in the **Tasks** folder. The first folder, "Notes Folder", will contain note items. The second

folder, "Contacts Folder", will contain contact items. The third folder, "Public Folder" will be a public folder. If the folders already exist, a message box will inform the user.

```
Sub AddFolders()
Dim myOlApp As New Outlook.Application
Dim myNameSpace As Outlook.NameSpace
Dim myFolder As Outlook.MAPIFolder
Dim myNotesFolder As Outlook.MAPIFolder
Dim myContactsFolder As Outlook.MAPIFolder
Dim myPublicFolder As Outlook.MAPIFolder
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", olFolderNo
Set myContactsFolder = myFolder.Folders.Add("Contacts Folder", olFo
Set myPublicFolder = myFolder.Folders.Add("Public Folder", olPublic
Exit Sub
ErrorHandler:
 MsgBox "This folder already exists!"
 Resume Next
End Sub
```

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

This VBA example gets the current **Contacts** folder and adds a new <u>ContactItem</u> object to it and sets some initial values in the fields based on another contact. To run this example without any error, replace 'Dan Wilson' with a valid contact name that exists in your Contacts folder.

```
Sub AddContact()
Dim myOlApp As New Outlook.Application
Dim myNamespace As Outlook.NameSpace
Dim myFolder As Outlook.MAPIFolder
Dim myItem As Outlook.ContactItem
Dim myOtherItem As Outlook.ContactItem
Set myNamespace = myOlApp.GetNamespace("MAPI")
Set myFolder = myNamespace.GetDefaultFolder(olFolderContacts)
Set myOtherItem = myFolder.Items("Dan Wilson")
Set myItem = myFolder.Items.Add
myItem.CompanyName = myOtherItem.CompanyName
myItem.BusinessAddress = myOtherItem.BusinessTelephoneNumbe
myItem.Display
```

End Sub

This VBA example adds a custom form to the default **Tasks** folder.

```
Sub AddForm()
Dim myOlApp As New outlook.Application
Dim myNamespace As outlook.NameSpace
Dim myItems As outlook.Items
Dim myFolder As outlook.MAPIFolder
Dim myItem As outlook.TaskItem
Set myNamespace = myOlApp.GetNamespace("MAPI")
Set myFolder = _______
myNamespace.GetDefaultFolder(olFolderTasks)
Set myItems = myFolder.Items
Set myItem = myItems.Add("IPM.Task.myTask")
End Sub
```

As it applies to the **OutlookBarGroups** collection.

This Visual Basic/VBA example adds a group named Marketing as the last group in the Shortcuts pane.

```
Sub AddGroup()
Dim myOlApp As New Outlook.Application
Dim myolBar As Outlook.OutlookBarPane
Set myolBar = myOlApp.ActiveExplorer.Panes.Item("OutlookBar")
myolBar.Contents.Groups.Add "Marketing", myolBar.Contents.Groups.Co
End Sub
```

If you use VBScript in an Outlook form, 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.Contents.Groups.Add "Marketing", myolBar.Contents.Groups.Cou
```

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.

```
Sub AddShortcut()
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", 1
End Sub
```

If you use VBScript in an Outlook form, you do not create the <u>Application</u> object. This example shows how to perform the same task using VBScript code.

```
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", 1
```

As it applies to the **Recipients** collection.

This VBA example creates a new mail message, uses the **Add** method to add 'Dan Wilson' as a **To** recipient, and displays the message. To run this example without errors, replace 'Dan Wilson' with a valid recipient name.

```
Sub CreateStatusReportToBoss()
Dim myOlApp As Outlook.Application
Dim myItem As Outlook.MailItem
Dim myRecipient As Outlook.Recipient
Set myOlApp = CreateObject("Outlook.Application")
Set myItem = myOlApp.CreateItem(olMailItem)
Set myRecipient = myItem.Recipients.Add("Dan Wilson")
myItem.Subject = "Status Report"
myItem.Display
End Sub
```

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

```
Set myItem = Application.CreateItem(0)
Set myRecipient = myItem.Recipients.Add("Dan Wilson")
myItem.Subject = "Status Report"
myItem.Display
```

As it applies to the **UserProperties** collection.

This VBA example creates a new <u>**ContactItem</u>** object and adds "LastDateSpokenWith" as a custom property.</u>

```
Sub AddUserProperty()
Dim myOlApp As New Outlook.Application
Dim myItem As Outlook.ContactItem
Dim myUserProperty As Outlook.UserProperty
Set myItem = myOlApp.CreateItem(olContactItem)
Set myUserProperty = myItem.UserProperties _
.Add("LastDateSpokenWith", olDateTime)
myItem.Display
End Sub
```

This VBA example creates a new **ContactItem** object and adds "Notes" as a user property. The **Value** is set by changing the **Value** property of the **UserProperty** object.

```
Sub AddUserProperty()
Dim myOlApp As New Outlook.Application
Dim myItem As Outlook.ContactItem
Dim myUserProperty As Outlook.UserProperty
Set myItem = myOlApp.CreateItem(olContactItem)
Set myUserProperty = myItem.UserProperties _
    .Add("Notes", olText)
myUserProperty.Value = "Neighbor"
myItem.Display
End Sub
```

As it applies to the Views object.

The following Visual Basic for Applications (VBA) example creates a new view called New Table and stores it in a variable called objNewView.

```
Sub CreateView()
'Creates a new view
```

End Sub

# **AddMember Method**

Adds a new member to the specified distribution list. The distribution list contains **<u>Recipient</u>** objects that represent valid e-mail addresses.

### expression.AddMember(Recipient)

*expression* Required. An expression that returns a **<u>DistListItem</u>** object.

*Recipient* Required. The recipient to be added to the list.

Use the <u>AddMembers</u> method to add multiple members to a given recipients list.

## **Example**

The following Microsoft Visual Basic/Visual Basic for Applications (VBA) example creates a new **DistributionList** object and adds a recipient to it. If the specified recipient is not valid, the **AddMember** method will fail. To run this example, replace 'Dan Wilson' with a valid recipient name.

```
Sub AddNewMember()
        'Adds a member to a new distribution list
        Dim olApp As Outlook.Application
        Dim objItem As Outlook.DistListItem
        Dim objMail As Outlook.MailItem
        Dim objRcpnt As Outlook.Recipient
        Set olApp = New Outlook.Application
        Set objMail = olApp.CreateItem(olMailItem)
        Set objItem = olApp.CreateItem(olDistributionListItem)
        'Create recipient for distlist
        Set objRcpnt = olApp.Session.CreateRecipient("Dan Wilson")
        objRcpnt.Resolve
        objItem.AddMember objRcpnt
        'Add note to list and display
        objItem.DLName = "Northwest Sales Manager"
        objItem.Body = "Regional Sales Manager - NorthWest"
        objItem.Save
        objItem.Display
```

End Sub

# **AddMembers Method**

Adds new members to a distribution list.

expression.AddMembers(Recipients)

*expression* Required. An expression that returns a **<u>DistListItem</u>** object.

*Recipients* Required <u>Recipients</u> object. The members to be added to the distribution list.

## Example

This Microsoft Visual Basic/Visual Basic for Applications (VBA) example creates a new distribution list and adds the current user and 'Dan Wilson' to the list. If the specified recipient is not valid, the **AddMember** method will fail. Therefore, to run this example, replace 'Dan Wilson' with a valid recipient name.

```
Sub AddNewMembers()
```

```
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 lis
myRecipients.Add myNameSpace.CurrentUser.Name
myRecipients.Add "Dan Wilson"
myRecipients.ResolveAll
myDistList.AddMembers myRecipients
myDistList.Save
myDistList.Display
```

```
End Sub
```

If you use Microsoft Visual Basic Scripting Edition (VBScript) in a Microsoft Outlook form, you do not create the <u>Application</u> 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
myRecipients.ResolveAll
myDistList.AddMembers myRecipients
myDistList.Save
```

myDistList.Display

# **AddPicture Method**

Adds a picture to a contact item.

### expression.AddPicture(Picture)

*expression* Required. An expression that returns a <u>ContactItem</u> object.

*Picture* Required. A string containing the complete path and filename of the picture to be added to the contact item.

If the contact item already has a picture attached to it, this method will overwrite the existing picture.

The picture can be an icon, GIF, JPEG, BMP, TIFF, WMF, EMF, or PNG file. Microsoft Outlook will automatically perform the necessary resizing of the picture.

## Example

The following Microsoft Visual Basic for Applications (VBA) example prompts the user to specify the name of a contact and the file name containing a picture of the contact, and then adds the picture to the contact item. If a picture already exists for the contact item, the example prompts the user to specify if the existing picture should be overwritten by the new file.

```
Sub AddPictureToAContact()
    Dim myOlApp As Outlook.Application
    Dim myNms As Outlook.NameSpace
    Dim myFolder As Outlook.MAPIFolder
    Dim myContactItem As Outlook.ContactItem
    Dim strName As String
    Dim strPath As String
    Dim strPrompt As String
    Set myOlApp = CreateObject("Outlook.Application")
    Set myNms = myOlApp.GetNamespace("MAPI")
    Set myFolder = myNms.GetDefaultFolder(olFolderContacts)
    strName = InputBox("Type the name of the contact: ")
    Set myContactItem = myFolder.Items(strName)
    If myContactItem.HasPicture = True Then
     strPrompt = MsgBox("The contact already has a picture associate
     If strPrompt = vbNo Then
     Exit Sub
     End If
    End If
    strPath = InputBox("Type the file name for the contact: ")
    myContactItem.AddPicture (strPath)
   myContactItem.Save
   myContactItem.Display
End Sub
```

# **AddStore Method**

Adds a Personal Folders (.pst) file to the current profile.

## expression.AddStore(Store)

*expression* Required. An expression that returns a **<u>NameSpace</u>** 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.

Use the **RemoveStore** method to remove a .pst that is already added to a profile.

## Example

This Microsoft Visual Basic/Visual Basic for Applications (VBA) example adds a new Personal Folders (.pst) file to the user's profile.

```
Sub CreatePST()
    Dim myOlApp As New Outlook.Application
    Dim myNameSpace As Outlook.NameSpace
    Set myNameSpace = myOlApp.GetNamespace("MAPI")
    myNameSpace.AddStore "c:\" & myNameSpace.CurrentUser & ".pst
End Sub
```

If you use Microsoft Visual Basic Scripting Edition (VBScript) in an Outlook form, you do not create the <u>Application</u> object. This example shows how to perform the same task using VBScript.

```
Sub CommandButton1_Click()
Set myNS = Application.GetNamespace("MAPI")
myNS.AddStore "c:\" & myNS.CurrentUser & ".pst"
End Sub
```



# **AddStoreEx Method**

Adds a Personal Folders file (.pst) in the specified format to the current profile.

## expression.AddStoreEx(Store, Type)

*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.

*Type* Required <u>OlStoreType</u> constant. The format in which the data file should be created.

OlStoreType can one of the following constants:

- olStoreDefault (1)
- olStoreUnicode (2)
- olStoreANSI (3)

## Remarks

Use the **olStoreUnicode** constant to add a new .pst file that has greater storage capacity for items and folders and supports multilingual Unicode data, to the user's profile. The **olStoreANSI** constant allows you to create .pst files that do not provide full support for multilingual Unicode data, but are compatible with earlier versions of Outlook. The **olStoreDefault** constant helps you create a .pst file in the default format that is compatible with the mailbox mode in which Outlook runs on the Microsoft Exchange Server.

This Microsoft Visual Basic/Visual Basic for Applications (VBA) example adds a new Personal Folders (.pst) file that has greater storage capacity for items and folders and supports Unicode to the user's profile.

```
Sub CreateUnicodePST()
   Dim myOlApp As New Outlook.Application
   Dim myNameSpace As Outlook.NameSpace
   Set myNameSpace = myOlApp.GetNamespace("MAPI")
   myNameSpace.AddStoreEx "c:\" & myNameSpace.CurrentUser & ".pst",
End Sub
```

If you use Microsoft Visual Basic Scripting Edition (VBScript) in an Outlook form, you do not create the **Application** object. This example shows how to perform the same task using VBScript.

```
Sub CreateUnicodePST()
        Set myNS = Application.GetNamespace("MAPI")
        myNS.AddStoreEx "c:\" & myNS.CurrentUser & ".pst", 2
End Sub
```

# **AddToFavorites Method**

Adds the current MAPI folder to the Microsoft Internet Explorer Favorites list.

**Note** The favorites are not accessible from the Microsoft Outlook user interface. However, you can still access them from Internet Explorer.

### expression.AddToFavorites(fNoUI, Name)

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

**fNoUI** Optional **Variant**. Specifies whether the Add Favorite dialog will be displayed. The default value is False, which displays the dialog. Specify True if you do not want to show the dialog to the user.

*Name* Optional **Variant**. Specifies the name of the favorite folder. The default value is the name of the folder.

The following Microsoft Visual Basic/Visual Basic for Applications (VBA) 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.

```
Sub FaveChange()
```

```
Dim appolApp As Outlook.Application
Dim nmsName As Outlook.NameSpace
Dim fldFolder As Outlook.MAPIFolder
Dim strName As String
Set appolApp = New 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("Type the name of the folder as it will appear in t
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 a Microsoft Exchange public folder to the public folder's Favorites folder.

### $expression. {\bf AddToPFF} avorites$

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

The following Visual Basic for Applications (VBA) example adds the public folder **GroupDiscussion** to the user's Favorites folder by using the **AddToPFFavorites** method. To run this example, you need to replace '**GroupDiscussion**' with a valid public folder name.

End Sub

# **AdvancedSearch Method**

Performs a search based on a specified Microsoft SQL Server search string and returns a <u>Search</u> object.

expression.AdvancedSearch(Scope, Filter, SearchSubFolders, Tag)

*expression* Required. An expression that returns an <u>Application</u> object.

*Scope* Required **String**. The scope of the search. For example, the name of a folder. It is recommended that the folder name is enclosed within single quotes. Otherwise, the search might not return correct results if the folder name contains special characters including Unicode characters.

*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. A maximum of 100 simultaneous searches can be performed by using the Microsoft Outlook user interface and the Outlook object model.

The **AdvancedSearch** method and related features in the Outlook object model do not create a Search Folder that will appear in the Outlook user interface. However, you can use the **Save** method of the Search object that is returned to create a Search Folder that will appear in the Search Folders list in the Outlook user interface.

The following Visual Basic for Applications (VBA) example searches the **Inbox** for items with subject equal to Test and displays the names of the senders of the e-mail items returned by the search. The AdvanceSearchComplete event procedure sets the boolean blnSearchComp to **True** when the the search is complete. This boolean variable is used by the TestAdvancedSearchComplete() procedure to determine when the search is complete. The sample code must be placed in a class module such as **ThisOutlookSession**, and the TestAdvancedSearchComplete() procedure can be called by Outlook.

```
Public blnSearchComp As Boolean
Private Sub Application_AdvancedSearchComplete(ByVal SearchObject As
        MsqBox "The AdvancedSearchComplete Event fired"
        blnSearchComp = True
End Sub
Sub TestAdvancedSearchComplete()
        Dim sch As Outlook.Search
        Dim rsts As Outlook.Results
        Dim i As Integer
        blnSearchComp = False
        Const strF As String = "urn:schemas:mailheader:subject = 'Te
        Const strS As String = "Inbox"
        Set sch = Application.AdvancedSearch(strS, strF)
        While blnSearchComp = False
                DoEvents
        Wend
        Set rsts = sch.Results
        For i = 1 To rsts.Count
                MsgBox rsts.Item(i).SenderName
        Next
End Sub
```

The following Microsoft Visual Basic/Visual Basic for Applications 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 have the Subject as 'Test'. The user's Inbox is specified as the scope of the search and the **SearchSubFolders** property is set to **True**. 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.

```
Public blnSearchComp As Boolean
Private Sub Application_AdvancedSearchComplete(ByVal SearchObject As
        MsqBox "The AdvancedSearchComplete Event fired for " & Searc
        blnSearchComp = True
End Sub
Sub TestAdvancedSearchComplete()
        Dim objSch As Outlook.Search
        Dim rsts As Outlook.Results
        Dim i As Integer
        blnSearchComp = False
        Const strF1 As String = "urn:schemas:mailheader:subject = 'T
        Const strS1 As String = "Inbox"
        Set objSch =
        Application.AdvancedSearch(Scope:=strS1, Filter:=strF1, Sear
        While blnSearchComp = False
                DoEvents
        Wend
        Set rsts = objSch.Results
        For i = 1 To rsts.Count
                MsgBox rsts.Item(i).SenderName
        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. Note that you need to use the AdvancedSearchComplete event to determine when the search is complete and work on the results.

```
Sub SearchForSubject()
'Search for all items with a certain subject
'in multiple folders
Dim objSch As Outlook.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

The following is another example that uses the **AdvancedSearch** method.

```
Public sch As Outlook.Search
Private Sub Application_AdvancedSearchComplete(ByVal SearchObject As
    Dim rsts As Outlook.Results
    If (SearchObject.Tag = "Search1") Then
        Set rsts = sch.Results
        MsgBox "Search1 returned " & rsts.Count & " items"
    End If
End Sub
Sub TestAdvancedSearchComplete()
    Dim rsts As Outlook.Results
    Dim i As Integer
    Const strF As String = "urn:schemas:mailheader:subject = 'Thnx'"
    Const strS As String = "Inbox"
    Set sch = Application.AdvancedSearch(strS, strF, , "Search1")
End Sub
```

# **Apply Method**

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

expression.Apply

*expression* 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.

This Microsoft Visual Basic/Visual Basic for Applications (VBA) 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.Combo1.Text
    globDirty = False
End Sub
```

The following Visual Basic for Applications (VBA) example creates a new view called **New Table** and applies it.

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

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

This Visual Basic for Applications (VBA) example uses <u>CreateItem</u> to create a simple task and delegate it as a task request to another user. To run this example, replace 'Dan Wilson' with a valid recipient name.

```
Sub AssignTask()
    Dim myOlApp As New Outlook.Application
    Dim myItem As Outlook.TaskItem
    Dim myDelegate As Outlook.Recipient
    Set MyItem = myOlApp.CreateItem(olTaskItem)
    MyItem.Assign
    Set myDelegate = MyItem.Recipients.Add("Dan Wilson")
    myDelegate.Resolve
    If myDelegate.Resolved Then
        myItem.Subject = "Prepare Agenda For Meeting"
        myItem.DueDate = Now + 30
        myItem.Display
        myItem.Send
    End If
Fad Sub
```

End Sub

If you use Microsoft Visual Basic Scripting Edition (VBScript) in a Microsoft Outlook form, you do not create the <u>Application</u> object, and you cannot use named constants. This example shows how to assign a task item using VBScript.

```
Set myItem = Application.CreateItem(3)
myItem.Assign
Set myDelegate = myItem.Recipients.Add("Dan Wilson")
myItem.Subject = "Prepare Agenda For Meeting"
myItem.DueDate = #9/20/03#
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 <u>TaskItem</u> object.

# **ClearConversationIndex Method**

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

### $expression. {\bf Clear Conversation Index}$

*expression* Required. An expression that returns a <u>MailItem</u> or <u>PostItem</u> 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 <u>AppointmentItem</u> or <u>TaskItem</u> object.



# **Close Method**

For the **Explorer** object, the **Close** method closes the explorer.

For an **Inspector** or Microsoft Outlook <u>item</u> 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 <u>OlInspectorClose</u>. 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.

This Visual Basic for Applications (VBA) example saves and closes the item displayed in the active inspector without prompting the user. To run this example, you need to have an item displayed in an inspector window.

```
Sub CloseItem()
    Dim myolapp As Outlook.Application
    Dim myinspector As Outlook.Inspector
    Dim myItem As Outlook.MailItem
    Set myolapp = CreateObject("Outlook.Application")
    Set myinspector = myolapp.ActiveInspector
    Set myItem = myinspector.CurrentItem
    myItem.Close olSave
End Sub
```

If you use Microsoft Visual Basic Scripting Edition (VBScript) in an Outlook form, you do not create the <u>Application</u> object, and you cannot use named constants. This example shows how to create a mail item, add a recipient, and close the item after prompting the user to save changes.

```
Set myItem = Application.CreateItem(0)
myItem.Recipients.Add "David Goodhand"
myItem.Close 2
```



# **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 <u>OlViewSaveOption</u>. 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.

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

The following Microsoft Visual Basic/Visual Basic for Applications (VBA) example creates a copy of a view called "New Table View" and saves it in the current folder. To run this example, you need to first create a view called 'New Table View' programmatically or by using the Microsoft Outlook user interface.

```
Sub CopyView()
'Copies a view
Dim olApp As Outlook.Application
Dim objViews As Outlook.Views
Dim objNewView As Outlook.View
Set olApp = New Outlook.Application
Set objViews = __
olApp.GetNamespace("MAPI").GetDefaultFolder(olFolderInbox).Views
'Create copy of View object
Set objNewView = objViews("New Table View").Copy(Name:="Table Vi
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 an e-mail message, sets the **Subject** to "Speeches", uses the **Copy** method to copy it, then moves the copy into a newly created e-mail folder named "Saved Mail" within the **Inbox** folder.

```
Sub CopyItem()
Dim myolApp As New Outlook.Application
Dim myNameSpace As Outlook.NameSpace
Dim myFolder As Outlook.MAPIFolder
Dim myNewFolder As Outlook.MAPIFolder
Dim myItem As Outlook.MailItem
```

```
Dim myCopiedItem As Outlook.MailItem
Set myolApp = CreateObject("Outlook.Application")
Set myNameSpace = myolApp.GetNamespace("MAPI")
Set myFolder = myNameSpace.GetDefaultFolder(olFolderInbox)
Set myNewFolder = myFolder.Folders.Add("Saved Mail", olFolde
Set myItem = myolApp.CreateItem(olMailItem)
myItem.Subject = "Speeches"
Set myCopiedItem = myItem.Copy
myCopiedItem.Move myNewFolder
```

End Sub

If you use Microsoft Visual Basic Scripting Edition (VBScript) in an Outlook form, you do not create the <u>Application</u> object and you cannot use named constants. This example shows how to perform the same task using VBScript code.

```
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 a Microsoft Outlook store and returns an **Object** representing the copied document.

expression.CopyFile(FilePath, DestFolderPath)

*expression* Required. An expression that returns an <u>Application</u> 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.

The following Visual Basic for Applications (VBA) example creates a Microsoft Excel worksheet called 'MyExcelDoc.xls' and then copies it from the user's hard drive to the user's **Inbox**.

```
Sub CopyFileSample()
Dim strPath As String
Dim ExcelApp As Object
Dim ExcelSheet As Object
Dim olApp As New Outlook.Application
Dim doc As Object
strPath = "C:\MyExcelDoc.xls"
Set ExcelApp = CreateObject("Excel.Application")
Set ExcelSheet = ExcelApp.Workbooks.Add
ExcelSheet.ActiveSheet.cells(1, 1).Value = 10
ExcelSheet.SaveAs strPath
ExcelApp.Quit
Set ExcelApp = Nothing
Set doc = olApp.CopyFile(strPath, "Inbox")
```

End Sub

# **CopyTo Method**

Copies the current folder in its entirety to the destination folder. Returns a **MAPIFolder** object that represents the new copy.

expression.CopyTo(DestinationFolder)

*expression* Required. An expression that returns a **MAPIFolder** object (source folder).

*DestinationFolder* Required **MAPIFolder** object (the destination folder for the copied folder).

This Visual Basic for Applications (VBA) example uses the **CopyTo** method to copy the default **Contacts** folder to the default **Inbox** folder.

```
Sub CopyFolder()
    Dim myOlApp As New Outlook.Application
    Dim myNameSpace As Outlook.NameSpace
    Dim myInboxFolder As Outlook.MAPIFolder
    Dim myContactsFolder As Outlook.MAPIFolder
    Dim myNewFolder As Outlook.MAPIFolder
    Set myNameSpace = myOlApp.GetNamespace("MAPI")
    Set myInboxFolder = myNameSpace.GetDefaultFolder(olFolderInb
    Set myContactsFolder = myNameSpace.GetDefaultFolder(olFolder
    Set myNewFolder = myContactsFolder.CopyTo(myInboxFolder)
End Sub
```

If you use Microsoft Visual Basic Scripting Edition (VBScript) in a Microsoft Outlook form, you do not create the <u>Application</u> object, and you cannot use named constants. This example shows how to perform the same task using VBScript code.

```
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 <u>item</u> and returns it. The **CreateItem** method can only create default Outlook items. To create new items using a custom form, use the <u>Add</u> method on the <u>Items</u> collection.

#### expression.CreateItem(ItemType)

*expression* Required. An expression that returns an <u>Application</u> object.

*ItemType* Required <u>OlItemType</u>. The Outlook item <u>Type</u> for the new item.

OlItemType can be one of these OlItemType constants.

olAppointmentItem olContactItem olDistributionListItem olJournalItem olMailItem olNoteItem olPostItem olTaskItem

The following Microsoft Visual Basic/Visual Basic for Applications (VBA) example creates a new **MailItem** object and sets the **BodyFormat** property to **olFormatHTML**. The Body text of the e-mail item will now appear in HTML format.

```
Sub CreateHTMLMail()
'Creates a new e-mail item and modifies its properties
Dim olApp As Outlook.Application
Dim objMail As Outlook.MailItem
Set olApp = Outlook.Application
'Create e-mail item
Set objMail = olApp.CreateItem(olMailItem)
With objMail
    'Set body format to HTML
    .BodyFormat = olFormatHTML
    .HTMLBody = "<HTML><H2>The body of this message will appear i
    .Display
End With
```

End Sub

If you use Microsoft Visual Basic Scripting Edition (VBScript) in a Microsoft Outlook form, you do not create the <u>Application</u> object, and you cannot use named constants. This example shows how to create a contact item in the default **Contacts** folder using VBScript code.

```
Sub CommandButton1_Click()
Set myNameSpace = Application.GetNameSpace("MAPI")
Set myFolder = myNameSpace.GetDefaultFolder(10)
Set myItem = Application.CreateItem(2)
myItem.Display
End Sub
```



# **CreateItemFromTemplate Method**

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

expression.CreateItemFromTemplate(TemplatePath, InFolder)

*expression* Required. An expression that returns an <u>Application</u> 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.

New items will always open in compose mode, as opposed to read mode, regardless of the mode in which the items were saved to disk.

This Visual Basic for Applications (VBA) example uses

**CreateItemFromTemplate** to create a new item from an Outlook template and then displays it. The CreateTemplate macro shows you how to create the template that is used in the first example. To avoid errors, replace 'Dan Wilson' with a valid name in your address book.

```
Sub CreateFromTemplate()
        Dim myOlApp As Outlook.Application
        Dim MyItem As Outlook.MailItem
        Set myOlApp = CreateObject("Outlook.Application")
        Set MyItem = myOlApp.CreateItemFromTemplate("C:\statusrep.of
        MyItem.Display
End Sub
Sub CreateTemplate()
        Dim myOlApp As Outlook.Application
        Dim MyItem As Outlook.MailItem
        Set myOlApp = CreateObject("Outlook.Application")
        Set MyItem = myOlApp.CreateItem(olMailItem)
        MyItem.Subject = "Status Report"
        MvItem.To = "Dan Wilson"
        MyItem.Display
        MyItem.SaveAs "C:\statusrep.oft", OlSaveAsType.olTemplate
End Sub
```

If you use Microsoft Visual Basic Scripting Edition (VBScript) in an Outlook form, you do not create the **Application** object. This example shows how to perform the same task using VBScript code.

```
Sub CommandButton1_Click()
Set myItem = Application.CreateItemFromTemplate
  ("C:\Program Files\Microsoft Office\Templates\Outlook\While You Wer
  myItem.Display
End Sub
```

The following Visual Basic for Applications (VBA) example shows how to use the optional **InFolder** parameter when calling the **CreateItemFromTemplate** method.



# **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 Microsoft Visual Basic Scripting Edition (VBScript) 1.0, which did not include a **CreateObject** method. **CreateObject** has been included in VBScript version 2.0 and later. This method should not be used to automate Microsoft 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 or Microsoft Visual Basic for Applications (VBA). These examples do *not* use the same **CreateObject** method that is implemented as part of the object model in Outlook.

#### expression.CreateObject(ObjectName)

*expression* Required. An expression that returns an <u>Application</u> object.

*ObjectName* Required **String**. The class name of the object to create. For information about valid class names, see <u>OLE Programmatic Identifiers</u>.

This VBScript example uses the **Open** event of the <u>item</u> to access Microsoft Internet Explorer and display the Web page.

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

This VBScript example uses the **Click** event of a **CommandButton** control on the <u>item</u> to access Microsoft Word and open a document in the root directory named "Resume.doc".

```
Sub CommandButton1_Click()
   Set Word = Application.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 **<u>NameSpace</u>** object.

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

This Visual Basic for Applications (VBA) example uses the **<u>GetSharedDefaultFolder</u>** method to resolve the <u>**Recipient**</u> object representing Dan Wilson, and then returns Dan's shared default **Calendar** folder. To run this example, replace 'Dan Wilson' with a valid recipient name and make sure the calendar is shared and you have permissions to view the calendar.

```
Sub ResolveName()
        Dim myOlApp As Outlook.Application
        Dim myNamespace As Outlook.NameSpace
        Dim myRecipient As Outlook.Recipient
        Dim CalendarFolder As Outlook.MAPIFolder
        Set myOlApp = CreateObject("Outlook.Application")
        Set myNamespace = myOlApp.GetNamespace("MAPI")
        Set myRecipient = myNamespace.CreateRecipient("Dan Wilson")
        myRecipient.Resolve
        If myRecipient.Resolved Then
                Call ShowCalendar(myNamespace, myRecipient)
        End If
End Sub
Sub ShowCalendar(myNamespace, myRecipient)
        Dim CalendarFolder As MAPIFolder
        Set CalendarFolder =
        myNamespace.GetSharedDefaultFolder _
        (myRecipient, olFolderCalendar)
        CalendarFolder.Display
End Sub
```

If you use Microsoft Visual Basic Scripting Edition (VBScript) in a Microsoft Outlook form, you do not create the <u>Application</u> object, and you cannot use named constants. This example shows how to perform the same task using VBScript code.

```
Sub CommandButton1_Click()
Set myNameSpace = Application.GetNameSpace("MAPI")
Set myRecipient = myNameSpace.CreateRecipient("Dan Wilson")
myRecipient.Resolve
If myRecipient.Resolved Then
    Set CalendarFolder = _______
    myNameSpace.GetSharedDefaultFolder _______
(myRecipient, 9)
```

CalendarFolder.Display End If End Sub

# **Delete Method**

Deletes an object from a collection.

#### expression.**Delete**

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

This Visual Basic for Applications (VBA) example uses the **Delete** method to delete the **PersonalTasks** folder within the **Tasks** folder. To run this example, you need to create a Tasks subfolder called PersonalTasks.

```
Sub DeleteTaskFolder()
        Dim myolApp As New Outlook.Application
        Dim oNamespace As Outlook.NameSpace
        Dim oFolder As Outlook.MAPIFolder
        Dim oOldFolder As Outlook.MAPIFolder
        Dim strPrompt As String
        Set oNamespace = myolApp.GetNamespace("MAPI")
        Set oFolder = oNamespace.GetDefaultFolder(olFolderTasks)
        Set oOldFolder = oFolder.Folders("PersonalTasks")
        'Prompt the user for confirmation
        strPrompt = "Are you sure you want to delete the folder?"
        If MsgBox(strPrompt, vbYesNo + vbQuestion) = vbYes Then
        oOldFolder.Delete
        MsgBox ("Folder deleted")
        End If
```

End Sub

If you use Microsoft Visual Basic Scripting Edition (VBScript) in a Microsoft Outlook form, you do not create the **Application** object, and you cannot use named constants. This example shows how to perform the same task using VBScript code.

```
Sub CommandButton1 Click()
 Set oNameSpace = Application.GetNameSpace("MAPI")
Set oFolder = oNameSpace.GetDefaultFolder(13)
Set oOldFolder = oFolder.Folders("PersonalTasks")
 'Prompt the user for confirmation
Dim strPrompt
 strPrompt = "Are you sure you want to delete the folder?"
 If MsgBox(strPrompt, vbYesNo + vbQuestion) = vbYes Then
 oOldFolder.Delete
 MsgBox("Folder deleted")
End If
```

End Sub

# **DeselectFolder Method**

If the explorer is currently displaying the **Calendar** folder simultaneously with another **Calendar** folder, the specified folder will be closed. Returns **Nothing**.

expression.DeselectFolder(MAPIFolder)

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

*MAPIFolder* Required. The MAPIFolder object representing the **Calendar** folder to be closed in the explorer.

The **DeselectFolder** method works only with **Calendar** folders.

If the folder is the only folder that is currently displayed in the explorer, the folder is not closed. Instead, Outlook will display an error.

This Microsoft Visual Basic for Applications (VBA) example closes a shared calendar that is displayed simultaneously with the current user's default calendar folder. To run this example without errors, replace Dan Wilson with a valid recipient name whose calendar is shared and whose calendar you have permissions to view. You also need to display the shared calendar in Shared Calendar view. To do this, you can run the DispCalendars procedure before running the CloseSharedCalendar procedure.

```
Sub CloseSharedCalendar()
    Dim myOlApp As Outlook.Application
    Dim myNms As Outlook.NameSpace
    Dim myRecipient As Outlook.Recipient
    Dim myExplorer As Outlook.Explorer
    Dim SharedFolder As Outlook.MAPIFolder
    Set myOlApp = CreateObject("Outlook.Application")
    Set myNms = myOlApp.GetNamespace("MAPI")
    Set myExplorer = myOlApp.ActiveExplorer
    Set myRecipient = myNms.CreateRecipient("Dan Wilson")
    Set SharedFolder = myNms.GetSharedDefaultFolder(myRecipient, olF
    If myExplorer.IsFolderSelected(SharedFolder) = True Then
        myExplorer. DeselectFolder SharedFolder
    End If
End Sub
Sub DispCalendars()
    Dim myOlApp As Outlook.Application
    Dim myNms As Outlook.NameSpace
    Dim myFolder As Outlook.MAPIFolder
    Dim myRecipient As Outlook.Recipient
    Dim myExplorer As Outlook.Explorer
    Dim SharedFolder As Outlook.MAPIFolder
    Set myOlApp = CreateObject("Outlook.Application")
    Set myNms = myOlApp.GetNamespace("MAPI")
    Set myFolder = myNms.GetDefaultFolder(olFolderCalendar)
    Set myExplorer = myOlApp.ActiveExplorer
    Set myExplorer.CurrentFolder = myFolder
    Set myRecipient = myNms.CreateRecipient("Dan Wilson")
```

```
Set SharedFolder = myNms.GetSharedDefaultFolder(myRecipient, olF
myExplorer.SelectFolder SharedFolder
End Sub
```

## **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 **<u>Name</u>** property is empty.

#### expression.Details(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.

# **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 **<u>NameSpace</u>** object.

*ContactItem* Optional Variant. The <u>ContactItem</u> object of the contact you want to dial.

The following Microsoft Visual Basic/Visual Basic for Applications (VBA) example opens the **New Call** dialog box.

Sub DialContact()
'Opens the New Call dialog
Dim olApp As Outlook.Application
Set olApp = Outlook.Application
olApp.GetNamespace("MAPI").Dial

End Sub

The following Visual Basic for Applications (VBA) example opens the **New Call** dialog box with the contact's information. To run this example, replace '**Jeff Smith**' with a valid contact name.

```
Sub DialContact()
'Opens the New Call dialog with the contact info
Dim olApp As Outlook.Application
Dim objContact As Outlook.ContactItem
Set olApp = New 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 **<u>Reminder</u>** object.

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

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

```
SSub DismissReminders()
'Dismisses any active reminders.
    Dim olApp As Outlook.Application
    Dim objRems As Outlook.Reminders
    Dim objRem As Outlook.Reminder
    Dim i As Integer
    Set olApp = New Outlook.Application
    Set objRems = olApp.Reminders
    For i = objRems.Count To 1 Step -1
        If objRems(i).IsVisible = True Then
            objRems(i).Dismiss
        End If
    Next
    Set olApp = Nothing
    Set objRems = Nothing
    Set objRem = Nothing
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 or 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 window modal. The default value is **False**.

If you attempt to open an "unsafe" file system object (or "freedoc" file) by using the Microsoft Outlook object model, you receive the E\_FAIL return code in the C or C++ programming languages. In Outlook 2000 and earlier, you could open an "unsafe" file system object by using the **Display** method.

This Visual Basic for Applications (VBA) 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.

```
Sub DisplayInbox()
        Dim myolApp As Outlook.Application
        Dim myNameSpace As Outlook.NameSpace
        Dim myFolder As Outlook.MAPIFolder
        Set myolApp = CreateObject("Outlook.Application")
        Set myNameSpace = myolApp.GetNamespace("MAPI")
        Set myFolder = myNameSpace.GetDefaultFolder(olFolderInbox)
        myFolder.Display
```

```
End Sub
```

If you use Microsoft Visual Basic Scripting Edition (VBScript) in an Outlook form, you do not create the **Application** object, and you cannot use named constants. This example shows how to perform the same task using VBScript code.

```
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 trying to display a specific item. If there are no items in the folder, a message box will be displayed to inform the user.

**Note** In Office Outlook 2003, the items in the **Items** collection object are not guaranteed to be in any particular order.

```
Sub DisplayFirstItem()
        Dim myolApp As Outlook.Application
        Dim myNameSpace As Outlook.NameSpace
        Dim myFolder As Outlook.MAPIFolder
        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."
```

End Sub



## **Execute Method**

Executes the action for the specified <u>item</u>. Returns the Microsoft Outlook item created by the action.

expression.Execute

*expression* Required. An expression that returns an <u>Action</u> object.

When you run a program that uses the Microsoft Outlook object model to call the Execute method, you receive a warning message. This warning message tells you that a program is trying to execute an action or verb on your behalf and asks if you want to allow that. The warning message contains both a Yes and a No button. However, the Yes button is not available until five seconds have passed since the warning message appeared. You can dismiss the warning message immediately if you click No.

This Visual Basic for Applications (VBA) example uses the **Execute** method to look through all the actions for the given e-mail message and executes the action called "Reply."

```
Sub SendReply()
        Dim myOlApp As Outlook.Application
        Dim myNameSpace As Outlook.NameSpace
        Dim MyItem As Outlook.MailItem
        Dim myItem2 As Outlook.MailItem
        Dim myAction As Outlook.Action
        Set myOlApp = CreateObject("Outlook.Application")
        Set myNameSpace = myOlApp.GetNamespace("MAPI")
        On Error GoTo ErrorHandler
        Set MyItem = myOlApp.ActiveInspector.CurrentItem
        For Each myAction In MyItem. Actions
                If myAction.Name = "Reply" Then
                        Set myItem2 = myAction.Execute
                        myItem2.Send
                        Exit For
                End If
        Next myAction
        Exit Sub
        ErrorHandler:
                MsgBox "There is no current item."
End Sub
```



# **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 <u>Items</u> object.

*Filter* Required **String**. The filter of the search.

The method will cause an error with the following properties:

Body	
Categories	LastFirstNoSpaceCompany
Children	LastFirstSpaceOnly
	LastFirstSpaceOnlyCompany
Class	LastFirstNoSpaceAndSuffix
Companies	MemberCount
CompanyLastFirstNoSpace	MemberCount
CompanyLastFirstSpaceOnly	NetMeetingAlias
	NetMeetingAutoStart
ContactNames	NetMeetingOrganizerAlias
Contacts	NetMeetingServer
ConversationIndex	-
DLName	NetMeetingType
Email1EntraID	RecurrenceState
Email1EntryID	ReplyRecipients
Email2EntryID	ReceivedByEntryID
Email3EntryID	
EntryID	RecevedOnBehalfOfEntryID
HTMLBody	ResponseState
-	Saved
IsOnlineMeeting	Sent
LastFirstAndSuffix	

LastFirstNoSpace	Submitted
AutoResolvedWinner	VotingOptions
BodyFormat	DownloadState
InternetCodePage	IsConflict
Permission	MeetingWorkspaceURL

**Creating Filters for the Find and Restrict Methods** 

The syntax for the filter varies depending on the type of field you are filtering on.

#### String (for Text fields)

When searching Text fields, you can use either an apostrophe (') or double quotation marks ("") to delimit the values that are part of the filter. For example, all of the following lines function correctly when the field is of type **String**:

sFilter = "[CompanyName] = 'Microsoft'"

sFilter = "[CompanyName] = ""Microsoft"""

sFilter = "[CompanyName] = " & Chr(34) & "Microsoft" & Chr(34)

**Note** If the search string contains a single quote character, escape the single quote character in the string with another single quote character. For example,

sFilter = "[Subject] = 'Can"t'"

Similarly, if the search string contains a double quote character, escape the double quote character in the string with another double quote character.

#### Date

Although dates and times are typically stored with a Date format, the **Find** and **Restrict** methods require that the date and time be converted to a string

representation. To make sure that the date is formatted as Microsoft Outlook expects, use the **Format** function. The following example creates a filter to find all contacts that have been modified after January 15, 1999 at 3:30 P.M.

sFilter = "[LastModificationTime] > "" & Format("1/15/99 3:30pm", "ddddd h:nn AMPM") & """

### **Boolean Operators**

**Boolean** operators, TRUE/FALSE, YES/NO, ON/OFF, and so on, should not be converted to a string. For example, to determine whether journaling is enabled for contacts, you can use this filter:

sFilter = "[Journal] = True"

**Note** If you use quotation marks as delimiters with **Boolean** fields, then an empty string will find items whose fields are False and all non-empty strings will find items whose fields are True.

### **Keywords (or Categories)**

The **Categories** field is of type keywords, which is designed to hold multiple values. When accessing it programmatically, the **Categories** field behaves like a Text field, and the string must match exactly. Values in the text string are separated by a comma and a space. This typically means that you cannot use the **Find** and **Restrict** methods on a keywords field if it contains more than one value. For example, if you have one contact in the Business category and one contact in the Business and Social categories, you cannot easily use the **Find** and **Restrict** methods to retrieve all items that are in the Business category. Instead, you can loop through all contacts in the folder and use the **Instr** function to test whether the string "Business" is contained within the entire keywords field.

**Note** A possible exception is if you limit the **Categories** field to two, or a low number of values. Then you can use the **Find** and **Restrict** methods with the OR logical operator to retrieve all Business contacts. For example (in pseudocode): "Business" OR "Business, Personal" OR "Personal, Business." Category strings are not case sensitive.

### Integer

You can search for **Integer** fields with or without quotation marks as delimiters. The following filters will find contacts that were created with Outlook 2000:

sFilter = "[OutlookInternalVersion] = 92711"

sFilter = "[OutlookInternalVersion] = '92711'"

#### **Using Variables as Part of the Filter**

As the **Restrict** method example illustrates, you can use values from variables as part of the filter. The following Microsoft Visual Basic Scripting Edition (VBScript) code sample illustrates syntax that uses variables as part of the filter.

sFullName = "Dan Wilson"

' This approach uses Chr(34) to delimit the value.

sFilter = "[FullName] = " & Chr(34) & sFullName & Chr(34)

' This approach uses double quotation marks to delimit the value.

sFilter = "[FullName] = """ & sFullName & """"

#### **Using Logical Operators as Part of the Filter**

Logical operators that are allowed are AND, OR, and NOT. The following are variations of the clause for the **Restrict** method, so you can specify multiple criteria.

OR: The following code returns all contact items that have either Business or Personal as their category.

sFilter = "[Categories] = 'Personal' Or [Categories] = 'Business'"

AND: The following code retrieves all personal contacts who work at Microsoft.

sFilter = "[Categories] = 'Personal' And [CompanyName] = 'Microsoft'"

NOT: The following code retrieves all personal contacts who don't work at Microsoft.

sFilter = "[Categories] = 'Personal' And Not([CompanyName] = 'Microsoft')"

### **Additional Notes**

If you are trying to use the **Find** or **Restrict** methods with user-defined fields, the fields must be defined in the folder, otherwise an error will occur. There is no way to perform a "contains" operation. For example, you cannot use **Find** or **Restrict** to search for items that have a particular word in the **Subject** field. Instead, you can use the **AdvancedSearch** method, or you can loop through all of the items in the folder and use the **InStr** function to perform a search within a field. You can use the **Find** and **Restrict** methods to search for items that begin within a certain range of characters. For example, to search for all contacts with a last name beginning with the letter M, use this filter:

sFilter = "[LastName] > 'LZZZ' And [LastName] < 'N'"

Find method as it applies to the UserProperties object.

Locates and returns a **<u>UserProperty</u>** 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 Boolean value that defines the search parameters. If Custom parameter is True, only custom user properties will be searched. The default value is True. To find a non custom property such as Subject, specify Custom parameter as False, otherwise will return Nothing.

This Visual Basic for Applications (VBA) example finds a custom property named "LastDateContacted" for the contact 'Jeff Smith'. To run this example, replace 'Jeff Smith' with a valid contact name and create a custom property called 'LastDateContacted' for the contact.

```
Sub FindContact()
'Finds and displays last contacted info for a contact
    Dim olApp As Outlook.Application
    Dim objContact As Outlook.ContactItem
    Dim objContacts As Outlook.MAPIFolder
    Dim objNameSpace As Outlook.NameSpace
    Dim objProperty As Outlook.UserProperty
    Set olApp = CreateObject("Outlook.Application")
    Set objNameSpace = olApp.GetNamespace("MAPI")
    Set objContacts = objNameSpace.GetDefaultFolder(olFolderContacts
    Set objContact = objContacts.Items.Find("[FileAs] = ""Smith, Jef
    If Not TypeName(objContact) = "Nothing" Then
        Set objProperty = objContact.UserProperties.Find("LastDateCo
        If TypeName(objProperty) <> "Nothing" Then
            MsgBox "Last Date Contacted: " & objProperty.Value
        End If
    Else
        MsgBox "Contact not found."
    End If
End Sub
```



# **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

*expression* Required. An expression that returns an <u>Items</u> collection object.

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

```
Sub DemoFindNext()
        Dim myOlApp As Outlook.Application
        Dim myNameSpace As Outlook.NameSpace
        Dim tdystart As Date
        Dim tdyend As Date
        Dim myAppointments As Outlook. Items
        Dim currentAppointment As Outlook.AppointmentItem
        Set myOlApp = CreateObject("Outlook.Application")
        Set myNameSpace = myOlApp.GetNamespace("MAPI")
        tdystart = VBA.Format(Now, "Short Date")
        tdyend = VBA.Format(Now + 1, "Short Date")
        Set myAppointments = myNameSpace.GetDefaultFolder(olFolderCa
        Set currentAppointment = myAppointments.Find("[Start] >= """
        While TypeName(currentAppointment) <> "Nothing"
       MsgBox currentAppointment.Subject
       Set currentAppointment = myAppointments.FindNext
Wend
End Sub
```

## **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.

This Visual Basic for Applications (VBA) example uses the **Remove** method to remove all attachments from a forwarded message before sending it on to Dan Wilson. To run this example, replace 'Dan Wilson' with a valid recipient name and keep a mail item that contains at least one attachment open in the active window.

```
Sub RemoveAttachmentBeforeForwarding()
    Dim myolApp As Outlook.Application
    Dim myinspector As Outlook. Inspector
    Dim myItem As Outlook.MailItem
    Dim myattachments As Outlook.Attachments
    Set myolApp = CreateObject("Outlook.Application")
    Set myinspector = myolApp.ActiveInspector
    If Not TypeName(myinspector) = "Nothing" Then
        Set myItem = myinspector.CurrentItem.Forward
        Set myattachments = myItem.Attachments
        While myattachments.Count > 0
               myattachments.Remove 1
        Wend
        myItem.Display
        myItem.Recipients.Add "Dan Wilson"
        myItem.Send
    Else
        MsgBox "There is no active inspector."
    End If
End Sub
```

# ForwardAsVcal Method

Forwards the **<u>AppointmentItem</u>** as a vCal; virtual calendar item. The **ForwardAsVcal** method returns a **<u>MailItem</u>** 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 *MinPerChar* minute, up to one month of information from the specified *Start* date).

If the optional argument *CompleteFormat* is omitted or **False**, then "free" is indicated by the character 0 and all other states by the character 1.

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

expression.FreeBusy(Start, MinPerChar, CompleteFormat)

*expression* Required. An expression that returns a **<u>Recipient</u>** object.

*Start* Required **Date**. The start date for the returned period of free/busy information.

*MinPerChar* Required Long. The number of minutes per character represented in the returned free/busy string.

*CompleteFormat* Optional Variant. True if the returned string should contain not only free/busy information, but also values for each character according to the OlBusyStatus constants: olBusy, olFree, olOutOfOffice, and olTentative.

This Visual Basic for Applications (VBA) 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. To run this example, you need to replace 'Nate Sun' with a valid recipient name.

```
Public Sub GetFreeBusyInfo()
    Dim myOlApp As New Outlook.Application
    Dim myNameSpace As Outlook.NameSpace
    Dim myRecipient As Outlook.Recipient
    Dim myFBInfo As String
    Set myNameSpace = myOlApp.GetNamespace("MAPI")
    Set myRecipient = myNameSpace.CreateRecipient("Nate Sun")
    On Error GoTo ErrorHandler
    myFBInfo = myRecipient.FreeBusy(#11/11/2003#, 60 * 24)
    MsgBox myFBInfo
    Exit Sub
    ErrorHandler:
        MsgBox "Cannot access the information. "
```

End Sub

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

Set myRecipient = myNameSpace.CreateRecipient("Nate Sun")
myFBInfo = myRecipient.FreeBusy(#8/1/03#, 60, True)

# **GetAssociatedAppointment Method**

Returns an **<u>AppointmentItem</u>** object that represents the appointment associated with the meeting request.

expression.GetAssociatedAppointment(AddToCalendar)

*expression* Required. An expression that returns a <u>MeetingItem</u> object.

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

This Visual Basic for Applications (VBA) example finds a <u>MeetingItem</u> in the default **Inbox** folder that has not been responded to yet and adds the associated appointment to the **Calendar** folder. It then responds to the sender by accepting the meeting.

```
Sub AcceptMeeting()
    Dim myOlApp As New Outlook.Application
    Dim myNameSpace As Outlook.NameSpace
    Dim myFolder As Outlook.MAPIFolder
    Dim myMtgReq As Outlook.MeetingItem
    Dim myAppt As Outlook.AppointmentItem
    Dim myMtg As Outlook.MeetingItem
    Set myNameSpace = myOlApp.GetNamespace("MAPI")
    Set myFolder = myNameSpace.GetDefaultFolder(olFolderInbox)
    Set myMtgReq = myFolder.Items.Find("[MessageClass] = 'IPM.Sc
    If TypeName(myMtgReq) <> "Nothing" Then
        Set myAppt = myMtgReq.GetAssociatedAppointment(True)
        Set myMtg = myAppt.Respond(olResponseAccepted, True)
        myMtg.Send
    End If
```

If you use Microsoft Visual Basic Scripting Edition (VBScript) in a Microsoft Outlook form, you do not create the <u>Application</u> object, and you cannot use named constants. This example shows how to obtain the associated appointment item for a meeting request using VBScript code.

```
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
```

## **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.

This Microsoft Visual Basic/Visual Basic for Applications (VBA) example accepts a <u>TaskRequestItem</u>, sending the response without displaying the inspector.

```
Sub AcceptTask()
        Dim myOlApp As New Outlook.Application
        Dim myNameSpace As Outlook.NameSpace
        Dim myTasks As Outlook.MAPIFolder
        Dim myNewTaskItem As Outlook.TaskItem
        Dim mytaskreqItem As Outlook.TaskRequestItem
        Dim myItem As Outlook.TaskItem
        Set myNameSpace = myOlApp.GetNamespace("MAPI")
        Set myTasks = myNameSpace.GetDefaultFolder(olFolderInbox)
        Set mytaskreqItem = myTasks.Items.Find("[Subject] = ""Meetin
        If Not TypeName(mytaskreqItem) = "Nothing" Then
                Set myNewTaskItem = mytaskregItem.GetAssociatedTask(
                Set myItem = myNewTaskItem.Respond(olTaskAccept, Tru
                myItem.Send
        End If
End Sub
```

If you use Microsoft Visual Basic Scripting Edition (VBScript) in a Microsoft Outlook form, you do not create the <u>Application</u> object, and you cannot use named constants. This example shows how to perform the same task using VBScript code.

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



# **GetDefaultFolder Method**

Returns a **MAPIFolder** object that represents the default folder of the requested type for the current profile, for example, obtains 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 **<u>NameSpace</u>** object.

*FolderType* Required <u>OlDefaultFolders</u>. The type of default folder to return.

OlDefaultFolders can be one of these OlDefaultFolders constants.

olFolderCalendar olFolderContacts olFolderDeletedItems olFolderDrafts olFolderInbox olFolderInbox olFolderJournal olFolderNotes olFolderOutbox olFolderOutbox olFolderSentMail olFolderTasks olPublicFoldersAllPublicFolders olFolderJunk

This Visual Basic for Applications (VBA) example uses the <u>CurrentFolder</u> property to change the displayed folder to the user's default **Calendar** folder.

```
Sub ChangeCurrentFolder()
    Dim myolApp As Outlook.Application
    Dim myNamespace As Outlook.NameSpace
    Set myolApp = CreateObject("Outlook.Application")
    Set myNamespace = myolApp.GetNamespace("MAPI")
    Set myolApp.ActiveExplorer.CurrentFolder = ______
    myNamespace.GetDefaultFolder(olFolderCalendar)
End Sub
```

End Sub

If you use Microsoft Visual Basic Scripting Edition (VBScript) in a Microsoft Outlook form, you do not create the <u>Application</u> object, and you cannot use named constants. This example shows how to perform the same task using VBScript code.

```
Set myNameSpace = Application.GetNameSpace("MAPI")
Set Application.ActiveExplorer.CurrentFolder = _
myNameSpace.GetDefaultFolder(9)
```

This VBA example returns the first folder in the Tasks Folders collection.

```
Sub DisplayATaskFolder()
Dim myolApp As Outlook.Application
Dim myNamespace As Outlook.NameSpace
Dim myTasks As Outlook.MAPIFolder
Dim myFolder As Outlook.MAPIFolder
Set myolApp = CreateObject("Outlook.Application")
Set myNamespace = myolApp.GetNamespace("MAPI")
Set myTasks = myNamespace.GetDefaultFolder(olFolderTasks)
Set myFolder = myTasks.Folders(1)
myFolder.Display
End Sub
```



## **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 **<u>Display</u>** 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 and the <u>Navigation Pane</u> on the left. 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**). This mode is essentially same as the **Normal** mode, in that it too displays the Navigation Pane on the left.

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).

This Visual Basic for Applications (VBA) example uses the **GetExplorer** method to return a new, inactive **Explorer** for the default Contacts folder, disables the user from customizing the commandbars, and then displays it in the default mode of **olFolderDisplayNormal**. Once the sub routine is run, the user will not be able to add or remove toolbar buttons or customize the menu options.

```
Sub DisableCustomize()
        Dim outApp As New Outlook.Application
        Dim nsp As Outlook.NameSpace
        Dim mpfContacts As Outlook.MAPIFolder
        Dim expContacts As Outlook.Explorer
        Set nsp = outApp.GetNamespace("MAPI")
        Set mpfContacts = nsp.GetDefaultFolder(olFolderContacts)
        Set expContacts = mpfContacts.GetExplorer
        expContacts.CommandBars.DisableCustomize = True
        expContacts.Activate
```

End Sub

If you use Microsoft Visual Basic Scripting Edition (VBScript) in an Outlook form, you do not create the **Application** object, and you cannot use named constants. This example shows how to display an explorer using VBScript.

```
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.

This Visual Basic for Applications (VBA) example uses the **GetFirst** method to locate the first folder in the **Contacts** folder and then copies the folder to the Test folder. Before running this example, you need to make sure the necessary folders exist in the default Contacts and Inbox folders.

```
Sub CopyItems()
    Dim myOlApp As New Outlook.Application
    Dim myNameSpace As Outlook.NameSpace
    Dim myDestFolder As Outlook.MAPIFolder
    Dim mySourceFolder As Outlook.MAPIFolder
    Dim myNewFolder As Outlook.MAPIFolder
    Set myNameSpace = myOlApp.GetNamespace("MAPI")
    Set myDestFolder = myNameSpace.GetDefaultFolder(olFolderInbc
    Set mySourceFolder = myNameSpace.GetDefaultFolder(olFolderCo
    Set myNewFolder = mySourceFolder.CopyTo(myDestFolder)
End Sub
```

If you use Microsoft Visual Basic Scripting Edition (VBScript) in a Microsoft Outlook form, you do not create the <u>Application</u> object, and you cannot use named constants. This example shows how to delete the first folder in the default **Tasks** folder using VBScript code.

# **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 <u>NameSpace</u> object.

*EntryIDFolder* Required **String**. The **<u>EntryID</u> of the folder**.

*EntryIDStore* Optional Variant. The <u>StoreID</u> for the folder.

This Visual Basic for Applications (VBA) example obtains the **EntryID** and **StoreID** for the default **Tasks** folder and then calls the **GetFolderFromID** method using these values to obtain the same folder. The folder is then displayed.

```
Sub GetWithID()
    Dim myOlApp As Outlook.Application
    Dim myFolder As Outlook.MAPIFolder
    Dim myEntryID As String
    Dim myStoreID As String
    Dim myNewFolder As Outlook.MAPIFolder
    Set myOlApp = CreateObject("Outlook.Application")
    Set myFolder = myOlApp.Session.GetDefaultFolder(olFolderTask
    myEntryID = myFolder.EntryID
    myStoreID = myFolder.StoreID
    Set myNewFolder = myOlApp.Session.GetFolderFromID(myEntryID,
    myNewFolder.Display
End Sub
```

```
End Sub
```

If you use Microsoft Visual Basic Scripting Edition (VBScript) in an Outlook form, you do not create the <u>Application</u> object, and you cannot use named constants. This example shows how to perform the same task using VBScript code.

```
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**.

#### expression.GetFreeBusy(Start, MinPerChar, CompleteFormat)

*expression* Required. The <u>AddressEntry</u> 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 <u>item</u> 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 **<u>NameSpace</u>** object.

*EntryIDItem* Required **String**. The **EntryID** of the item.

*EntryIDStore* Optional Variant. The <u>StoreID</u> for the folder. *EntryIDStore* usually must be provided when retrieving an item based on its MAPI IDs.

## Remarks

For more information about Entry IDs, see the **EntryID** property.

## **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.

The following Visual Basic for Applications example searches the subfolders of **Inbox** for a folder called **MyPersonalEmails** and displays a message to the user. If you do not have a subfolder called **MyPersonalEmails** in your **Inbox** folder, the example will display nothing.

```
Sub TestGetLast()
    Dim outApp As New Outlook.Application
    Dim nsp As Outlook.NameSpace
    Dim mpf As Outlook.MAPIFolder
    Dim mpfSubFolder As Outlook.MAPIFolder
    Dim flds As Outlook.Folders
    Dim idx As Integer
    Set nsp = outApp.GetNamespace("MAPI")
    Set mpf = nsp.GetDefaultFolder(olFolderInbox)
    Set flds = mpf.Folders
    Set mpfSubFolder = flds.GetLast
    Do While Not mpfSubFolder Is Nothing
        If mpfSubFolder.Name = "MyPersonalEmails" Then
            MsqBox "The folder was found."
            Exit Do
        End If
        Set mpfSubFolder = flds.GetPrevious
    Loop
```

End Sub

## **GetMember Method**

Returns a **<u>Recipient</u>** object representing a member in a distribution list.

#### expression.GetMember(Index)

*expression* Required. An expression that returns a **<u>DistListItem</u>** object.

*Index* Required **Long**. The index number of the member to be retrieved.

### Remarks

If a program tries to reference any type of recipient information by using the Microsoft Outlook object model, a dialog box is displayed that asks you to confirm access to this information. You can allow access to the Address Book or recipient information for up to ten minutes after you receive the dialog box. This allows features, such as mobile device synchronization, to be completed.

You receive the confirmation dialog box when a solution tries to programmatically access the **GetMember** method.

This Microsoft Visual Basic/Visual Basic for Applications (VBA) example locates every distribution list in the default **Contacts** folder and determines whether the list contains the current user.

```
Sub DisplayYourDLNames()
        Dim myOlApp As New Outlook.Application
        Dim myNameSpace As Outlook.NameSpace
        Dim myFolder As Outlook.MAPIFolder
        Dim myDistList As Outlook.DistListItem
        Dim myFolderItems As Outlook.Items
        Dim x As Integer
        Dim y As Integer
        Dim iCount As Integer
        Set myNameSpace = myOlApp.GetNamespace("MAPI")
        Set myFolder = myNameSpace.GetDefaultFolder(olFolderContacts
        Set myFolderItems = myFolder.Items
        iCount = myFolderItems.Count
        For x = 1 To iCount
                If TypeName(myFolderItems.Item(x)) = "DistListItem"
                Set myDistList = myFolderItems.Item(x)
                For y = 1 To myDistList.MemberCount
                        If myDistList.GetMember(y).Name = myNameSpac
                                MsgBox "Your are a member of " & myD
                        End If
                        Next y
                End If
        Next x
```

```
End Sub
```

If you use Microsoft Visual Basic Scripting Edition (VBScript) in an Outlook form, you do not create the <u>Application</u> object, and you cannot use named constants. This example shows how to perform the same task using VBScript code.

```
Set myNameSpace = Application.GetNamespace("MAPI")
Set myFolder = myNameSpace.GetDefaultFolder(10)
Set myFolderItems = myFolder.Items
iCount = myFolderItems.Count
For x = 1 To iCount
If TypeName(myFolderItems.Item(x)) = "DistListItem" Then
        Set myDistList = myFolderItems.Item(x)
        For y = 1 To myDistList.MemberCount
```

## **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.

The following Visual Basic for Applications example searches the subfolders of **Inbox** for a folder called **MyPersonalEmails** and displays a message to the user. If you do not have a subfolder called **MyPersonalEmails** in your **Inbox** folder, the example will display nothing.

```
Sub TestGetNext()
    Dim outApp As New Outlook.Application
    Dim nsp As Outlook.NameSpace
    Dim mpf As Outlook.MAPIFolder
    Dim mpfSubFolder As Outlook.MAPIFolder
    Dim flds As Outlook.Folders
    Dim idx As Integer
    Set nsp = outApp.GetNamespace("MAPI")
    Set mpf = nsp.GetDefaultFolder(olFolderInbox)
    Set flds = mpf.Folders
    Set mpfSubFolder = flds.GetFirst
    Do While Not mpfSubFolder Is Nothing
        If mpfSubFolder.Name = "MyPersonalEmails" Then
            MsqBox "The folder was found."
            Exit Do
        End If
        Set mpfSubFolder = flds.GetNext
```

Loop

End Sub

## **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.

This Visual Basic for Applications (VBA) 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 Microsoft Visual Basic Scripting Edition (VBScript), see the Note at the end of the example.

```
Public Sub cmdExample()
    Dim myOlApp As Outlook.Application
    Dim myApptItem As Outlook.AppointmentItem
    Dim myRecurrPatt As Outlook.RecurrencePattern
    Dim myNamespace As Outlook.NameSpace
    Dim myFolder As Outlook.MAPIFolder
    Dim myItems As Outlook. Items
    Dim myDate As Date
    Dim myOddApptItem As Outlook.AppointmentItem
    Dim saveSubject As String
    Dim newDate As Date
    Dim myException As Outlook.Exception
    Set myOlApp = New Outlook.Application
    Set myApptItem = myOlApp.CreateItem(olAppointmentItem)
   myApptItem.Start = #2/2/2003 3:00:00 PM#
   myApptItem.End = #2/2/2003 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 2/2/03 and ends on 2/2/04
```

```
'and save it.
    Set myRecurrPatt = myApptItem.GetRecurrencePattern
   myRecurrPatt.RecurrenceType = olRecursDaily
   myRecurrPatt.PatternStartDate = #2/2/2003#
    myRecurrPatt.PatternEndDate = #2/2/2004#
    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 3/12/03.
    myDate = #3/12/2003 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 = #3/12/2003 3:30:00 PM#
    mvOddApptItem.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.
    MsqBox myException.OriginalDate & ": " & saveSubject
    'Display the current date, time, and subject
    'for this exception.
    MsqBox 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 **<u>PropertyPage</u>** 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.

This Microsoft Visual Basic/Visual Basic for Applications (VBA) 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.

The following Visual Basic for Applications example searches the subfolders of **Inbox** for a folder called **MyPersonalEmails** and displays a message to the user. If you do not have a subfolder called **MyPersonalEmails** in your **Inbox** folder, the example will display nothing.

```
Sub TestGetPrevious()
    Dim outApp As New Outlook.Application
    Dim nsp As Outlook.NameSpace
    Dim mpf As Outlook.MAPIFolder
    Dim mpfSubFolder As Outlook.MAPIFolder
    Dim flds As Outlook.Folders
    Dim idx As Integer
    Set nsp = outApp.GetNamespace("MAPI")
    Set mpf = nsp.GetDefaultFolder(olFolderInbox)
    Set flds = mpf.Folders
    Set mpfSubFolder = flds.GetLast
    Do While Not mpfSubFolder Is Nothing
        If mpfSubFolder.Name = "MyPersonalEmails" Then
            MsqBox "The folder was found."
            Exit Do
        End If
        Set mpfSubFolder = flds.GetPrevious
    Loop
```

End Sub

# 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 **<u>NameSpace</u>** object.

*EntryID* Required **String**. The **EntryID** of the recipient.

### Remarks

If a program tries to reference any type of recipient information by using the Outlook object model, a dialog box is displayed that asks you to confirm access to this information. You can allow access to the Address Book or recipient information for up to ten minutes after you receive the dialog box. This allows features, such as mobile device synchronization, to be completed.

You receive the confirmation dialog box when a solution tries to programmatically access the **GetRecipientFromID** method.

This Visual Basic for Applications (VBA) example gets the entry ID of the first recipient in the item in the **Inbox** folder with subject 'Test', obtains the recipient from the entry ID, and displays the recipient name. To run this example without any errors, make sure there is a mail item with subject 'Test' in the Inbox. The example may also fail if there are other types of items with subject 'Test' other than mail item in the **Inbox**.

```
Public Sub GetFromID()
    Dim nsp As Outlook.NameSpace
    Dim mpfInbox As Outlook.MAPIFolder
    Dim mail As Outlook.MailItem
    Dim rcp As Outlook.Recipient
    Dim rcp1 As Outlook.Recipient
    Dim strEntryId As String
    Set nsp = Application.GetNamespace("MAPI")
    Set mpfInbox = nsp.GetDefaultFolder(olFolderInbox)
    Set mail = mpfInbox.Items("Test")
    Set rcp = mail.Recipients.Item(1)
    strEntryId = rcp.EntryID
    Set rcp1 = nsp.GetRecipientFromID(strEntryId)
    MsgBox rcp1.Name
```

End Sub

## 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 <u>AppointmentItem</u> or <u>TaskItem</u> object.

This Visual Basic for Applications (VBA) 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 Microsoft Visual Basic Scripting Edition (VBScript), see the Note at the end of the example.

```
Public Sub cmdExample()
    Dim myOlApp As Outlook.Application
    Dim myApptItem As Outlook.AppointmentItem
    Dim myRecurrPatt As Outlook.RecurrencePattern
    Dim myNamespace As Outlook.NameSpace
    Dim myFolder As Outlook.MAPIFolder
    Dim myItems As Outlook. Items
    Dim myDate As Date
    Dim myOddApptItem As Outlook.AppointmentItem
    Dim saveSubject As String
    Dim newDate As Date
    Dim myException As Outlook.Exception
    Set myOlApp = New Outlook.Application
    Set myApptItem = myOlApp.CreateItem(olAppointmentItem)
   myApptItem.Start = #2/2/2003 3:00:00 PM#
   myApptItem.End = #2/2/2003 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 2/2/03 and ends on 2/2/04
```

```
'and save it.
    Set myRecurrPatt = myApptItem.GetRecurrencePattern
   myRecurrPatt.RecurrenceType = olRecursDaily
   myRecurrPatt.PatternStartDate = #2/2/2003#
    myRecurrPatt.PatternEndDate = #2/2/2004#
    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 3/12/03.
    myDate = #3/12/2003 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 = #3/12/2003 3:30:00 PM#
    mvOddApptItem.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.
    MsqBox myException.OriginalDate & ": " & saveSubject
    'Display the current date, time, and subject
    'for this exception.
    MsqBox 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)
```



# GetSharedDefaultFolder Method

Returns a **MAPIFolder** object that represents the specified default folder for the specified user. This method is used in a 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 <u>NameSpace</u> object.

*Recipient* Required <u>Recipient</u> object. The owner of the folder. The **Recipient** object must be resolved.

*FolderType* Required <u>OlDefaultFolders</u> object. The type of folder.

OlDefaultFolders can be one of these OlDefaultFolders constants. olFolderCalendar olFolderContacts olFolderDrafts olFolderInbox olFolderJournal olFolderNotes olFolderSharedRoot olFolderTasks olFolderJunk

### Remarks

Microsoft Outlook does not allow you to open the following folders using the **GetSharedDefaultFolder** method. Therefore, the following constants cannot be used with this method:

- Deleted Items olFolderDeletedItems
- OutBox **olFolderOutbox**
- Sent Items olFolderSentMail
- All Public Folders olPublicFoldersAllPublicFolders

This Visual Basic for Applications (VBA) example uses the **GetSharedDefaultFolder** method to resolve the **Recipient** object representing Dan Wilson, and then returns Dan's shared default **Calendar** folder.

```
Sub ResolveName()
        Dim myOlApp As Outlook.Application
        Dim myNamespace As Outlook.NameSpace
        Dim myRecipient As Outlook.Recipient
        Dim CalendarFolder As Outlook.MAPIFolder
        Set myOlApp = CreateObject("Outlook.Application")
        Set myNamespace = myOlApp.GetNamespace("MAPI")
        Set myRecipient = myNamespace.CreateRecipient("Dan Wilson")
        myRecipient.Resolve
        If myRecipient.Resolved Then
                Call ShowCalendar(myNamespace, myRecipient)
        End If
End Sub
Sub ShowCalendar(myNamespace, myRecipient)
        Dim CalendarFolder As Outlook.MAPIFolder
        Set CalendarFolder =
        myNamespace.GetSharedDefaultFolder _
        (myRecipient, olFolderCalendar)
        CalendarFolder.Display
End Sub
```

If you use Microsoft Visual Basic Scripting Edition (VBScript) in an Outlook form, you do not create the <u>Application</u> object, and you cannot use named constants. This example shows how to perform the same task using VBScript code.

```
Set myNameSpace = Application.GetNameSpace("MAPI")
Set myRecipient = myNameSpace.CreateRecipient("Dan Wilson")
myRecipient.Resolve
If myRecipient.Resolved Then
    Set CalendarFolder = _______myNameSpace.GetSharedDefaultFolder ______(myRecipient, 9)
    CalendarFolder.Display
End If
```

## **HideFormPage Method**

Hides a form page in the inspector.

expression.HideFormPage(PageName)

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

*PageName* Required **String**. The display name of the page to be hidden.

This Visual Basic for Applications (VBA) example uses <u>HideFormPage</u> to hide the "General" page of a newly-created <u>ContactItem</u> and displays the item.

```
Sub HidePage()
    Dim myOlApp As Outlook.Application
    Dim MyItem As Outlook.ContactItem
    Dim myPages As Outlook.Pages
    Dim myinspector As Outlook.Inspector
    Set myOlApp = CreateObject("Outlook.Application")
    Set MyItem = myOlApp.CreateItem(olContactItem)
    Set myPages = MyItem.GetInspector.ModifiedFormPages
    myPages.Add "General"
    Set myinspector = myOlApp.ActiveInspector
    myinspector.HideFormPage "General"
    MyItem.Display
Fad Sub
```

```
End Sub
```

If you use Microsoft Visual Basic Scripting Edition (VBScript) in a Microsoft Outlook form, you do not create the <u>Application</u> object, and you cannot use named constants. This example shows how to perform the same task using VBScript code.

```
Set myItem = Application.CreateItem(2)
Set myInspector = myItem.GetInspector
Set myPages = myInspector.ModifiedFormPages
myPages.Add "General"
myInspector.HideFormPage"General"
myItem.Display
```

# **IsFolderSelected Method**

Returns a **Boolean** that determines if the folder is currently displayed in the explorer.

expression. IsFolderSelected(MAPIFolder)

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

*MAPIFolder* Required. The MAPIFolder object representing the folder that is displayed in the explorer.

This Microsoft Visual Basic for Applications (VBA) example closes a shared calendar that is displayed simultaneously with the current user's default **Calendar** folder. To run this example without errors, replace Dan Wilson with a valid recipient name whose calendar is shared and whose calendar you have permissions to view. You also need to display the shared calendar in Shared Calendar view. To do this, you can run the DispCalendar's procedure before running the CloseSharedCalendar procedure.

```
Sub CloseSharedCalendar()
    Dim myOlApp As Outlook.Application
    Dim myNms As Outlook.NameSpace
    Dim myRecipient As Outlook.Recipient
    Dim myExplorer As Outlook.Explorer
    Dim SharedFolder As Outlook.MAPIFolder
    Set myOlApp = CreateObject("Outlook.Application")
    Set myNms = myOlApp.GetNamespace("MAPI")
    Set myExplorer = myOlApp.ActiveExplorer
    Set myRecipient = myNms.CreateRecipient("Dan Wilson")
    Set SharedFolder = myNms.GetSharedDefaultFolder(myRecipient, olF
    If myExplorer.IsFolderSelected(SharedFolder) = True Then
        myExplorer.DeselectFolder SharedFolder
    End If
End Sub
Sub DispCalendars()
    Dim myOlApp As Outlook.Application
    Dim myNms As Outlook.NameSpace
    Dim myFolder As Outlook.MAPIFolder
    Dim myRecipient As Outlook.Recipient
    Dim myExplorer As Outlook.Explorer
    Dim SharedFolder As Outlook.MAPIFolder
    Set myOlApp = CreateObject("Outlook.Application")
    Set myNms = myOlApp.GetNamespace("MAPI")
    Set myFolder = myNms.GetDefaultFolder(olFolderCalendar)
    Set myExplorer = myOlApp.ActiveExplorer
    Set myExplorer.CurrentFolder = myFolder
    Set myRecipient = myNms.CreateRecipient("Dan Wilson")
```

```
Set SharedFolder = myNms.GetSharedDefaultFolder(myRecipient, olF
myExplorer.SelectFolder SharedFolder
End Sub
```



## **IsPaneVisible Method**

Returns a **Boolean** indicating whether a specific explorer pane is visible. Returns **True** if the specified pane is displayed in the explorer.

**Note** You can also use the <u>Visible</u> property of the <u>OutlookBarPane</u> object to determine whether the <u>Shortcuts pane</u> 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 (2) olOutlookBar (1) olPreview (3) olNavigationPane (4)

This Microsoft Visual Basic/Visual Basic for Applications (VBA) sample uses the <u>IsPaneVisible</u> method to determine whether the preview pane is visible and uses the <u>ShowPane</u> method to display it if it is not visible. Use the **olNavigationPane** constant to hide or display the <u>Navigation Pane</u>.

```
Sub HidePreviewPane()
   Dim myOlApp As New Outlook.Application
   Dim myOlExp As Outlook.Explorer
   Set myOlExp = myOlApp.ActiveExplorer
   If myOlExp.IsPaneVisible(olPreview) = False Then
        myOlExp.ShowPane olPreview, True
   End If
   Set myOlApp = Nothing
   Set myOlExp = Nothing
End Sub
```

If you use Microsoft Visual Basic Scripting Edition (VBScript) in a Microsoft Outlook form, you do not create the <u>Application</u> object, and you cannot use named constants. This example shows how to perform the same task using VBScript code.

```
Set myOlExp = Application.ActiveExplorer
If myOlExp.IsPaneVisible(3) = False Then
    myOlExp.ShowPane 3,True
    Set myOlExp = Nothing
End If
```

# IsSearchSynchronous Method

Returns a **Boolean** indicating if a search will be synchronous or asynchronous. Read-only.

**Note** If the search is synchronous, the **AdvancedSearch** method will not return until the search has completed. Conversely, if the search is asynchronous, the **AdvancedSearch** method will immediately return. 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.

#### expression. IsSearchSynchronous(LookInFolders)

*expression* Required. An expression that returns an <u>Application</u> object.

*LookInFolders* Required **String.** The path name of the folders that the search will search through. It is recommended that the folder name is enclosed within single quotes.

# 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**.

expression.IsWordMail

*expression* 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.

Collection

**Object Returned** 

**Actions** Action **AddressEntries AddressEntry AddressLists AddressList Attachments Attachment Exceptions Exception Explorers Explorer Folders MAPIFolder Inspectors Inspector** Outlook item **Items** Links Link **OutlookBarGroups OutlookBarGroup OutlookBarShortcuts OutlookBarShortcut** Page **Pages Panes** Pane **PropertyPages PropertyPage Selection** Outlook item **SyncObject SyncObjects Recipients Recipient UserProperties UserProperty** 

All other Microsoft Outlook A generic **Object** representing a single object in collections 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.

The following Microsoft Visual Basic/Visual Basic for Applications (VBA) 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. To run this example without errors, select items of type MailItem only. Other types such as **ReportItem** do not have the **SenderName** property and will cause an error.

```
Sub GetSelectedItems()
        Dim myOlApp As New Outlook.Application
        Dim myOlExp As Outlook.Explorer
        Dim myOlSel As Outlook.Selection
        Dim MsgTxt As String
        Dim x As Integer
        MsgTxt = "You have selected items from: "
        Set myOlExp = myOlApp.ActiveExplorer
        Set myOlSel = myOlExp.Selection
        For x = 1 To myOlSel.Count
                MsgTxt = MsgTxt & myOlSel.Item(x).SenderName & ";"
        Next x
        MsgBox MsgTxt
```

End Sub

The following example adds the public folder **Internal** 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 Outlook.MAPIFolder
        Set olapp = Outlook.Application
        Set objFolder = olapp.Session.GetDefaultFolder(olPublicFolde
        objFolder.AddToPFFavorites
End Sub
```



# **Logoff Method**

Logs the user off from the current MAPI <u>session</u>.

expression.Logoff

*expression* An expression that returns a **<u>NameSpace</u>** object.



# **Logon Method**

Logs the user on to MAPI, obtaining a MAPI <u>session</u>.

#### expression.Logon(Profile, Password, ShowDialog, NewSession)

*expression* An expression that returns a **<u>NameSpace</u>** object.

*Profile* Optional **Variant**. The MAPI profile name, as a **String**, to use for the session.

*Password* Optional **Variant**. The password (if any), as a **String**, associated with the profile. This parameter exists only for backwards compatibility and for security reasons, it is not recommended for use. Microsoft Oultook will prompt the user to specify a password in most system configurations. This is your logon password and should not be confused with PST passwords.

*ShowDialog* Optional **Variant**. **True** to display the MAPI logon dialog box to allow the user to select a MAPI profile.

*NewSession* Optional **Variant**. **True** to create a new Outlook session. Since multiple sessions cannot be created in Outlook, this parameter should be specified as True only if a session does not already exist.

This Microsoft Visual Basic example uses the **Logon** method to log on to a new session, displaying the dialog box to verify the profile name and enter password.

```
Sub StartOutlook()
Dim myOlApp As Outlook.Application
Dim myNameSpace As Outlook.NameSpace
Set myOlApp = CreateObject("Outlook.Application")
Set myNameSpace = myOlApp.GetNamespace("MAPI")
myNameSpace.Logon "LatestProfile", , True, True
End Sub
```

# **MarkComplete Method**

Marks the task as completed. Sets **<u>PercentComplete</u>** to "100%", <u>**Complete**</u> to **True**, and <u>**DateCompleted**</u> to the current date.

expression.MarkComplete

*expression* Required. An expression that returns a **<u>TaskItem</u>** object.



## **Move Method**

Moves a Microsoft Outlook <u>item</u> 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 <u>MAPIFolder</u> object. The destination folder.

This Visual Basic for Applications (VBA) example uses **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 Dan Wilson and uses the **Move** method to move all e-mail messages sent by Dan Wilson from the default **Inbox** folder to the Personal Mail folder. To run this example without any errors, replace 'Dan Wilson' with a vaid sender name and make sure there's a folder under Inbox called 'Personal Mail'. Note that **myItem** is declared as type **Object** so that it can represent all types of Outlook items including meeting request and task request items.

```
Sub MoveItems()
```

```
Dim myOlApp As New Outlook.Application
Dim myNameSpace As Outlook.NameSpace
Dim myInbox As Outlook.MAPIFolder
Dim myDestFolder As Outlook.MAPIFolder
Dim myItems As Outlook.Items
Dim myItem As Object
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] = 'Dan Wilson'")
While TypeName(myItem) <> "Nothing"
myItem.Move myDestFolder
Set myItem = myItems.FindNext
Wend
```

```
End Sub
```

If you use Microsoft Visual Basic Scripting Edition (VBScript) in a Microsoft Outlook form, you do not create the **Application** object, and you cannot use named constants. This example shows how to perform the same task using VBScript code.

```
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] = 'Dan Wilson'")
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.

TThis Visual Basic for Applications (VBA) example uses the **MoveTo** method to move the **My Test Contacts** folder in the default **Contacts** folder to the Inbox folder.

```
Sub MoveFolder()
Dim myOlApp As New Outlook.Application
Dim myNameSpace As Outlook.NameSpace
Dim myFolder As Outlook.MAPIFolder
Dim myNewFolder As Outlook.MAPIFolder
Set myNameSpace = myOlApp.GetNamespace("MAPI")
Set myFolder = myNameSpace.GetDefaultFolder(olFolderContacts)
Set myNewFolder = myFolder.Folders.Add("My Test Contacts")
myNewFolder.MoveTo myNameSpace.GetDefaultFolder(olFolderInbox)
End Sub
```

If you use Microsoft Visual Basic Scripting Edition (VBScript) in a Microsoft Outlook form, you do not create the <u>Application</u> object, and you cannot use named constants. This example shows how to move a new folder created in the **Contacts** folder to the default **Inbox** folder using VBScript code.

```
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 **<u>PropertyPageSite</u>** object.

This Microsoft Visual Basic/Visual Basic for Applications (VBA) example shows how to call the **OnStatusChange** method to notify 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
   E
```

# **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.



# **PrintOut Method**

Prints the Outlook <u>item</u> 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 <u>form</u> 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 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 <u>OlFormRegistry</u>. 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.

This Visual Basic for Applications (VBA) 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.

```
Sub PublishToFolder()
Dim myOlApp As New Outlook.Application
Dim myNamespace As Outlook.NameSpace
Dim myFolder As Outlook.MAPIFolder
Dim myItem As Outlook.ContactItem
Dim myForm As Outlook.FormDescription
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
End Sub
```

This VBA 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, on the **File** menu, point to **New**, and click **Choose Form**. In the **Look in** box, click **Personal Forms Library**. To open your new form, double-click **Interview Scheduler**.

```
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 <u>session</u> 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 <u>Application</u> 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.

As it applies to the **Actions**, **Attachments**, **Folders**, **Items**, **ItemProperties**, **Pages**, **Recipients**, and **UserProperties** objects.

This Visual Basic for Applications (VBA) example uses the **Remove** method to remove all attachments from a forwarded message before sending it on to Dan Wilson. Before running this example, replace 'Dan Wilson' with a valid recipient name.

```
Sub RemoveAttachmentBeforeForwarding()
    Dim myolApp As Outlook.Application
    Dim myinspector As Outlook. Inspector
    Dim myItem As Outlook.MailItem
    Dim myattachments As Outlook.Attachments
    Set myolApp = CreateObject("Outlook.Application")
    Set myinspector = myolApp.ActiveInspector
    If Not TypeName(myinspector) = "Nothing" Then
        Set myItem = myinspector.CurrentItem.Forward
        Set myattachments = myItem.Attachments
        While myattachments.Count > 0
               myattachments. Remove 1
        Wend
        myItem.Display
        myItem.Recipients.Add "Dan Wilson"
        myItem.Send
    Else
        MsgBox "There is no active inspector."
    End If
End Sub
```

As it applies to the Links, OutlookBarGroups, OutlookBarShortcuts, PropertyPages, Reminders, and Views objects.

The following example removes a <u>View</u> object from the <u>Views</u> collection.

```
Sub DeleteView()
'Deletes a view from the collection
Dim olApp As Outlook.Application
```

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

End Sub

## **RemoveMember Method**

Removes an individual member from a given distribution list.

#### expression.RemoveMember(Recipient)

*expression* Required. An expression that returns a **<u>DistListItem</u>** object.

*Recipient* Required **Recipient**. The **Recipient** to be removed from the distribution list.

The following Microsoft Visual Basic/Visual Basic for Applications (VBA) example removes a member from the distribution list called Group List. The **RemoveMember** method will fail if the specified recipient is not valid. Before running the example, create or make sure a distribution list called 'Group List' exists in your default Contacts folder.

```
Sub RemoveRec()
'Remove a recipient from the list, and displays new list.
    Dim olApp As Outlook.Application
    Dim objDstList As Outlook.DistListItem
    Dim objName As Outlook.NameSpace
    Dim objRcpnt As Outlook.Recipient
    Dim objMail As Outlook.MailItem
    Set olApp = New Outlook.Application
    Set objName = olApp.GetNamespace("MAPI")
    Set objDstList = objName.GetDefaultFolder(olFolderContacts).Item
    Set objMail = olApp.CreateItem(olMailItem)
    Set objRcpnt = objMail.Recipients.Add(Name:="someone@example.com
    obiRcont.Resolve
    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 **<u>DistListItem</u>** object.

*Recipients* Required <u>Recipients</u>. The members to be removed from the distribution list.

The following Microsoft Visual Basic/Visual Basic for Applications (VBA) example removes two members from the distribution list called Group List. The **RemoveMembers** method will fail if the specified recipients are not valid. Before running the example, create or make sure a distribution list called 'Group List' exists in your default Contacts folder.

```
Sub RemoveRecs()
'Remove a recipient from the list and displays new list.
    Dim olApp As Outlook.Application
    Dim objDstList As Outlook.DistListItem
    Dim objName As Outlook.NameSpace
    Dim objRcpnt As Outlook.Recipient
    Dim objRcpnt2 As Outlook.Recipient
    Dim objMail As Outlook.MailItem
    Dim objRcpnts As Outlook.Recipients
    Set olApp = New Outlook.Application
    Set objName = olApp.GetNamespace("MAPI")
    Set objDstList = objName.GetDefaultFolder(olFolderContacts).Item
    Set objMail = olApp.CreateItem(olMailItem)
    Set objRcpnts = objMail.Recipients
    Set objRcpnt = objRcpnts.Add(Name:="someone@example.com")
    Set objRcpnt2 = objRcpnts.Add(Name:="someone@example.org")
    objRcpnts.ResolveAll
    objDstList.RemoveMembers objRcpnts
    objDstList.Display
    objDstList.Body = "Last Modified: " & Now
```

End Sub

# **RemovePicture Method**

Removes a picture from a **Contacts** item. Returns **Nothing**.

#### expression.RemovePicture

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

The following Microsoft Visual Basic for Applications (VBA) example prompts the user to specify a name of a contact and removes the picture from the contact item. If a picture does not exist for the contact, the example displays a message to the user.

```
Sub RemovePictureFromContact()
    Dim myOlApp As Outlook.Application
    Dim myNms As Outlook.NameSpace
    Dim myFolder As Outlook.MAPIFolder
    Dim myContactItem As Outlook.ContactItem
    Dim strName As String
    Dim strPath As String
    Dim strPrompt As String
    Set myOlApp = CreateObject("Outlook.Application")
    Set myNms = myOlApp.GetNamespace("MAPI")
    Set myFolder = myNms.GetDefaultFolder(olFolderContacts)
    strName = InputBox("Type the name of the contact: ")
    Set myContactItem = myFolder.Items(strName)
    If myContactItem.HasPicture = False Then
        MsgBox "The contact does not have a picture associated with
    Else
        myContactItem.RemovePicture
        myContactItem.Save
        myContactItem.Display
     End If
```

End Sub

## **RemoveStore Method**

Removes a Personal Folders file (.pst) from the current MAPI profile or session.

#### expression.RemoveStore(Folder)

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

*Folder* Required **MAPIFolder** object. The Personal Folders file (.pst) to be deleted from the list.

## Remarks

This method removes a store only from the Microsoft Outlook user interface. You cannot remove a store from the main mailbox on the server or from a user's hard disk using the Outlook object model.

The following Microsoft Visual Basic/Visual Basic for Applications (VBA) examples removes a folder called Personal Folders from the list of folders.

```
Sub RemovePST()
    Dim objOL As New Outlook.Application
    Dim objName As Outlook.NameSpace
    Dim objFolder As Outlook.MAPIFolder
    Set objName = objOL.GetNamespace("MAPI")
    Set objFolder = objName.Folders.Item("Personal Folders")
    'Prompt the user for confirmation
    Dim strPrompt As String
    strPrompt = "Are you sure you want to remove the Personal Fo
    If MsgBox(strPrompt, vbYesNo + vbQuestion) = vbYes Then
        objName.RemoveStore objFolder
    End If
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

*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.

The following Microsoft Visual Basic/Visual Basic for Applications (VBA) 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.

End Sub

# **ResetColumns Method**

The **ResetColumns** method clears the properties that have been cached with the <u>SetColumns</u> 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 **<u>Recipient</u>** 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.

## Remarks

When you run a program that uses the Microsoft Outlook object model to call the **Resolve** method, you receive a warning message. This warning message tells you that a program is trying to access the Address Book on your behalf and asks if you want to allow this.

This Visual Basic for Applications (VBA) example uses <u>CreateItem</u> to create a simple task and delegate it as a task request to another user. Before running this example, replace 'Dan Wilson' with a valid recipient name.

End Sub

# **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.

### Remarks

When you run a program that uses the Microsoft Outlook object model to call the **ResolveAll** method, you receive a warning message. This warning message tells you that a program is trying to access the Address Book on your behalf and asks if you want to allow this.

This Visual Basic for Applications (VBA) example uses the **ResolveAll** method to attempt to resolve all recipients and, if unsuccessful, displays a message box for each unresolved recipient.

```
Sub CheckRecipients()
 Dim myOlApp As New Outlook.Application
 Dim MyItem As Outlook.MailItem
 Dim myRecipients As Outlook.Recipients
 Dim myRecipient As Outlook.Recipient
 Set myOlApp = CreateObject("Outlook.Application")
 Set myItem = myOlApp.CreateItem(olMailItem)
 Set myRecipients = myItem.Recipients
 myRecipients.Add("Aaron Con")
myRecipients.Add("Nate Sun")
 myRecipients.Add("Dan Wilson")
 If Not myRecipients. ResolveAll Then
    For Each myRecipient In myRecipients
        If Not myRecipient.Resolved Then
            MsgBox myRecipient.Name
        End If
     Next
 End If
End Sub
```

If you use Microsoft Visual Basic Scripting Edition (VBScript) within an Outlook form, you do not create the <u>Application</u> object, and you cannot use named constants. This example shows how to perform the same task using VBScript code in a **CommandButton Click** event.

```
Sub CommandButton1_Click()
Set myItem = Application.CreateItem(0)
Set myRecipients = myItem.Recipients
myRecipients.Add("Aaron Con")
myRecipients.Add("Nate Sun")
myRecipients.Add("Dan Wilson")
If Not myRecipients.ResolveAll Then
For Each myRecipient In myRecipients
If Not myRecipient.Resolved Then
MsgBox myRecipient.Name
End If
Next
End If
```

End Sub



# **Respond Method**

Responds to a meeting request for the <u>AppointmentItem</u> object or a task request for the <u>TaskItem</u> object.

expression.Respond(Response, fNoUI, fAdditionalTextDialog)

*expression* Required. An expression that returns an **AppointmentItem** or **TaskItem** object.

*Response* Required <u>OlMeetingResponse</u>. 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:

### Remarks

When you run a program that uses the Microsoft Outlook object model to call the **Respond** method, you receive a warning message. This warning message tells you that a program is trying to send an item on your behalf and asks if you want to allow this.

fNoUI, fAdditionalTextDialog	Result
	For <b>AppointmentItem</b> and <b>TaskItem</b> :
True, True	Response item is returned with no user interface. To send the response, you must call the <u>Send</u> method.
True, False False, True	For <b>AppointmentItem</b> and <b>TaskItem</b> :
	Same result as with <b>True, True</b> .
	For <b>AppointmentItem</b> :
	Prompts user to Send or Edit before sending the response.
	For <b>TaskItem</b> :
	If the <b>Display</b> method has been called, the user prompt appears. Otherwise, the item is sent without prompting and the resulting item is nothing.
False, False	For <b>AppointmentItem</b> :
	New response item appears in the user interface, but no prompt is displayed.
	For <b>TaskItem</b> :
	Does nothing.

This Visual Basic for Applications (VBA) example finds a <u>MeetingItem</u> in the default **Inbox** folder and adds the associated appointment to the **Calendar** folder. It then responds to the sender by accepting the meeting.

```
Sub AcceptMeeting()
    Dim myOlApp As New Outlook.Application
    Dim myNameSpace As Outlook.NameSpace
    Dim myFolder As Outlook.MAPIFolder
    Dim myMtgReq As Outlook.MeetingItem
    Dim myAppt As Outlook.AppointmentItem
    Dim myMtg As Outlook.MeetingItem
    Set myNameSpace = myOlApp.GetNamespace("MAPI")
    Set myFolder = myNameSpace.GetDefaultFolder(olFolderInbox)
    Set myMtgReq = myFolder.Items.Find("[MessageClass] = 'IPM.Sc
    If TypeName(myMtgReq) <> "Nothing" Then
        Set myAppt = myMtgReq.GetAssociatedAppointment(True)
        Set myMtg = myAppt.Respond(olResponseAccepted, True)
        myMtg.Send
    End If
```

If you use Microsoft Visual Basic Scripting Edition (VBScript) in an Outlook form, you do not create the <u>Application</u> object, and you cannot use named constants. This example shows how to perform the same task using VBScript in a **CommandButton Click** event.

```
Sub CommandButton1_Click()
Set myNamespace = Application.GetNamespace("MAPI")
Set myFolder = myNamespace.GetDefaultFolder(6)
Set myMtgReq = myFolder.Items.Find("[MessageClass] = 'IPM.Schedule.
If TypeName(myMtgReq) <> "Nothing" Then
    Set myAppt = myMtgReq.GetAssociatedAppointment(True)
    myAppt.Respond 3, False, True
Else
    MsgBox "You have no meeting requests."
End If
End Sub
```

## **Restrict Method**

Applies a filter to the **Items** collection, returning a new collection containing all of the items from the original that 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 a 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.Restrict(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	LastFirstNoSpaceCompany
Children	LastFirstSpaceOnly
Class	LastFirstSpaceOnlyCompany
	LastFirstNoSpaceAndSuffix
Companies	MemberCount
CompanyLastFirstNoSpace	NetMeetingAlias
CompanyLastFirstSpaceOnly	- -
ContactNames	NetMeetingAutoStart
Contacts	NetMeetingOrganizerAlias
ConversationIndex	NetMeetingServer
	NetMeetingType
DLName	RecurrenceState
Email1EntryID	ReplyRecipients
Email2EntryID	ReceivedByEntryID
Email3EntryID	
EntryID	RecevedOnBehalfOfEntryID
HTMLBody	ResponseState
-	Saved
IsOnlineMeeting	Sent

LastFirstAndSuffix	Submitted
LastFirstNoSpace	VotingOptions
AutoResolvedWinner	DownloadState
BodyFormat	IsConflict
InternetCodePage	MeetingWorkspaceURL
<b></b>	

Permission

**Creating Filters for the Find and Restrict Methods** 

The syntax for the filter varies depending on the type of field you are filtering on.

#### String (for Text fields)

When searching Text fields, you can use either an apostrophe ('), or double quotation marks (""), to delimit the values that are part of the filter. For example, all of the following lines function correctly when the field is of type **String**:

sFilter = "[CompanyName] = 'Microsoft'"

sFilter = "[CompanyName] = ""Microsoft"""

sFilter = "[CompanyName] = " & Chr(34) & "Microsoft" & Chr(34)

**Note** If the search string contains a single quote character, escape the single quote character in the string with another single quote character. For example,

sFilter = "[Subject] = 'Can"t'"

Similarly, if the search string contains a double quote character, escape the double quote character in the string with another double quote character.

#### Date

Although dates and times are typically stored with a Date format, the **Find** and

**Restrict** methods require that the date and time be converted to a string representation. To make sure that the date is formatted as Microsoft Outlook expects, use the **Format** function. The following example creates a filter to find all contacts that have been modified after January 15, 1999 at 3:30 P.M.

sFilter = "[LastModificationTime] > "" & Format("1/15/99 3:30pm", "ddddd h:nn AMPM") & """

#### **Boolean Operators**

**Boolean** operators, TRUE/FALSE, YES/NO, ON/OFF, and so on, should not be converted to a string. For example, to determine whether journaling is enabled for contacts, you can use this filter:

sFilter = "[Journal] = True"

**Note** If you use quotation marks as delimiters with **Boolean** fields, then an empty string will find items whose fields are **False** and all non-empty strings will find items whose fields are True.

#### **Keywords (or Categories)**

The **Categories** field is of type keywords, which is designed to hold multiple values. When accessing it programmatically, the **Categories** field behaves like a Text field, and the string must match exactly. Values in the text string are separated by a comma and a space. This typically means that you cannot use the **Find** and **Restrict** methods on a keywords field if it contains more than one value. For example, if you have one contact in the Business category and one contact in the Business and Social categories, you cannot easily use the **Find** and **Restrict** methods to retrieve all items that are in the Business category. Instead, you can loop through all contacts in the folder and use the **Instr** function to test whether the string "Business" is contained within the entire keywords field.

**Note** A possible exception is if you limit the Categories field to two, or a low number of values. Then you can use the **Find** and **Restrict** methods with the OR logical operator to retrieve all Business contacts. For example (in pseudocode): "Business" OR "Business, Personal" OR "Personal, Business." Category strings are not case sensitive.

#### Integer

You can search for **Integer** fields with, or without quotation marks as delimiters. The following filters will find contacts that were created using Outlook 2000:

sFilter = "[OutlookInternalVersion] = 92711"

sFilter = "[OutlookInternalVersion] = '92711'"

#### **Using Variables as Part of the Filter**

As the **Restrict** method example illustrates, you can use values from variables as part of the filter. The following Microsoft Visual Basic Scripting Edition (VBScript) code sample illustrates syntax that uses variables as part of the filter.

sFullName = "Dan Wilson"

' This approach uses Chr(34) to delimit the value.

sFilter = "[FullName] = " & Chr(34) & sFullName & Chr(34)

' This approach uses double quotation marks to delimit the value.

sFilter = "[FullName] = """ & sFullName & """"

#### Using Logical Operators as Part of the Filter

Logical operators that are allowed are AND, OR, and NOT. The following are variations of the clause for the **Restrict** method so you can specify multiple criteria.

OR: The following code returns all contact items that have either Business or Personal as their category.

sFilter = "[Categories] = 'Personal' Or [Categories] = 'Business'"

AND: The following code retrieves all personal contacts who work at Microsoft.

sFilter = "[Categories] = 'Personal' And [CompanyName] = 'Microsoft'"

NOT: The following code retrieves all personal contacts who don't work at Microsoft.

sFilter = "[Categories] = 'Personal' And Not([CompanyName] = 'Microsoft')"

#### **Additional Notes**

If you are trying to use the **Find** or **Restrict** methods with user-defined fields, the fields must be defined in the folder, otherwise an error will occur. There is no way to perform a "contains" operation. For example, you cannot use **Find** or **Restrict** to search for items that have a particular word in the **Subject** field. Instead, you can use the **AdvancedSearch** method, or you can loop through all of the items in the folder and use the **InStr** function to perform a search within a field. You can use the **Find** and **Restrict** methods to search for items that begin within a certain range of characters. For example, to search for all contacts with a last name beginning with the letter M, use this filter:

sFilter = "[LastName] > 'LZZZ' And [LastName] < 'N'"

This Visual Basic for Applications (VBA) example uses the **Restrict** method to get all Inbox items of **Business** category and moves them to the **Business** folder. To run this example, create or make sure a subfolder called 'Business' exists under Inbox.

```
Sub MoveItems()
        Dim myOlApp As Outlook.Application
        Dim myNamespace As Outlook.NameSpace
        Dim myFolder As Outlook.MAPIFolder
        Dim myItems As Outlook. Items
        Dim myRestrictItems As Outlook.Items
        Dim myItem As Outlook.MailItem
        Set myOlApp = CreateObject("Outlook.Application")
        Set myNamespace = myOlApp.GetNamespace("MAPI")
        Set myFolder =
        myNamespace.GetDefaultFolder(olFolderInbox)
        Set myItems = myFolder.Items
        Set myRestrictItems = myItems.Restrict("[Categories] = 'Busi
        For i = myRestrictItems.Count To 1 Step -1
                myRestrictItems(i).Move myFolder.Folders("Business")
        Next
```

End Sub

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

```
Sub CommandButton1_Click()
 Set myNameSpace = Application.GetNameSpace("MAPI")
Set myFolder = myNameSpace.GetDefaultFolder(6)
Set myItems = myFolder.Items
Set myRestrictItems = myItems.Restrict _
                                              ("[Categories] = 'Busi
For i = myRestrictItems.Count To 1 Step -1
                myRestrictItems(i).Move myFolder.Folders("Business")
Next
End Sub
```

This Visual Basic for Applications example uses the **Restrict** method to apply a filter to contact items based on the item's **LastModificationTime** property.

```
Public Sub ContactDateCheck()
    Dim myOlApp As Outlook.Application
    Dim myNamespace As Outlook.NameSpace
    Dim myContacts As Outlook. Items
    Dim myItems As Outlook. Items
    Dim myItem As Object
    Set myOlApp = CreateObject("Outlook.Application")
    Set myNamespace = myOlApp.GetNamespace("MAPI")
    Set myContacts = myNamespace.GetDefaultFolder(olFolderContacts).
    Set myItems = myContacts.Restrict("[LastModificationTime] > '01/
    For Each myItem In myItems
        If (myItem.Class = olContact) Then
            MsgBox myItem.FullName & ": " & myItem.LastModificationT
        End If
    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 ContactDateCheck2()
        Dim myOlApp As Outlook.Application
        Dim myNamespace As Outlook.NameSpace
        Dim myContacts As Outlook. Items
        Dim myItem As Outlook.Object
        Dim DateStart As Date
        Dim DateToCheck As String
        Dim myRestrictItems As Outlook.Items
        Set myOlApp = CreateObject("Outlook.Application")
        Set myNameSpace = myOlApp.GetNamespace("MAPI")
        Set myContacts = myNameSpace.GetDefaultFolder(olFolderContac
        DateStart = #01/1/2003#
        DateToCheck = "[LastModificationTime] >= """ & DateStart & "
        Set myRestrictItems = myContacts.Restrict(DateToCheck)
        For Each myItem In myRestrictItems
                 If (myItem.Class = olContact) Then
            MsqBox myItem.FullName & ": " & myItem.LastModificationT
        End If
        Next
End Sub
```



# **Save Method**

As it applies to the **Search** object

Saves the search results to a <u>Search Folder</u>.

expression.Save(SchFldrName)

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

*SchFldrName* Required. A string that represents the Search Folder name.

As it applies to the **View** object

Saves the view, or saves the changes to a view.

expression.Save

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

As it applies to the other objects in the Applies To list

Saves the Microsoft Outlook <u>item</u> 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.

### Remarks

The **Save** method displays an error if a Search Folder with the same name already exists.

As it applies to the **Search** object

The following Microsoft Visual Basic for Applications (VBA) example searches the **Inbox** for items with **Subject** line equal to 'Test' and saves the results in a Search Folder. The AdvanceSearchComplete event procedure sets the Boolean blnSearchComp to **True** when the the search is complete. This Boolean variable is used by the TestAdvancedSearchComplete() procedure to determine when the search is complete. The sample code must be placed in a class module such as **ThisOutlookSession**, and the TestAdvancedSearchComplete() procedure must be called before the event procedure can be called by Outlook.

```
Public blnSearchComp As Boolean
Private Sub Application_AdvancedSearchComplete(ByVal SearchObject As
        MsgBox "The AdvancedSearchComplete Event fired"
        blnSearchComp = True
End Sub
Sub TestAdvancedSearchComplete()
        Dim sch As Outlook.Search
        Dim rsts As Outlook.Results
        Dim i As Integer
        blnSearchComp = False
        Const strF As String = "urn:schemas:mailheader:subject = 'Te
        Const strS As String = "Inbox"
        Set sch = Application.AdvancedSearch(strS, strF)
        While blnSearchComp = False
                DoEvents
        Wend
        sch.Save("Subject Test")
End Sub
```

```
As it applies to the View object
```

The following VBA example creates a new view called New Table and applies it.

```
Sub CreateView()
```

```
'Creates a new view
```

End Sub

As it applies to the other objects in the Applies To list

This Microsoft Visual Basic/Visual Basic for Applications (VBA) example creates an appointment item and sets the **ReminderSet** property before saving it.

```
Sub AddAppointment()
    Dim OutApp As Outlook.Application
    Dim apti As Outlook.AppointmentItem
    Set OutApp = CreateObject("Outlook.Application")
    Set apti = OutApp.CreateItem(olAppointmentItem)
    apti.Subject = "Car Servicing"
    apti.Start = DateAdd("n", 16, Now)
    apti.End = DateAdd("n", 60, apti.Start)
    apti.ReminderSet = True
    apti.ReminderMinutesBeforeStart = 60
    apti.Save
End Sub
```

End Sub

If you use Microsoft Visual Basic Scripting Edition (VBScript) in an Outlook form, you do not create the <u>Application</u> object, and you cannot use named constants. This example shows how to perform the same task using VBScript.

```
Sub CommandButton1_Click()
Set apti = Application.CreateItem(1)
apti.Subject = "Car Servicing"
apti.Start = DateAdd("n", 16, Now)
apti.End = DateAdd("n", 60, apti.Start)
apti.ReminderSet = True
apti.ReminderMinutesBeforeStart = 60
```

apti.**Save** End Sub



### **SaveAs Method**

Saves the Microsoft Outlook <u>item</u> to the specified path and in the format of the specified file type. If the file type is not specified, the MSG format (.msg) 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: olHTML, olMSG, olRTF, olTemplate, olDoc, olTXT, olVCal, olVCard, olICal, or olMSGUnicode.

### Remarks

When you use the **SaveAs** method to save items to the file system, you receive an "address book" warning message. This includes all types of items, whether or not the items have attachments or active content. This change has been made so that someone cannot programmatically save items to a file and then parse the file to retrieve e-mail addresses.

Also note that even though **olDoc** is a valid **OlSaveAsType** constant, messages in HTML format cannot be saved in Document format, and the **olDoc** constant works only if Microsoft Word is set up as the default email editor.

This Visual Basic for Applications (VBA) example uses the **SaveAs** method to save the currently open item as a text file in the C:\ folder, using the subject as the file name. To run this example, make sure a mail item in plain text format is open in the active window.

```
Sub SaveAsTXT()
        Dim myItem As Outlook. 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
                'Prompt the user for confirmation
                Dim strPrompt As String
                strPrompt = "Are you sure you want to save the item?
                If MsqBox(strPrompt, vbYesNo + vbQuestion) = vbYes T
                        objItem.SaveAs "C:\" & strname & ".txt", ol
                End If
        Else
                MsgBox "There is no current active inspector."
        End If
```

```
End Sub
```

If you use Microsoft Visual Basic Scripting Edition (VBScript) in an Outlook form, you do not create the **Application** object, and you cannot use named constants. This example shows how to save the current item using VBScript code.

```
Sub CommandButton1 Click()
  Const OLTXT = 0
  strname = Item.Subject
        'Prompt the user for confirmation
        Dim strPrompt
         strPrompt = "Are you sure you want to save the item? If a f
        If MsgBox(strPrompt, vbYesNo + vbQuestion) = vbYes Then
                Item.SaveAs "C:\" & strname & ".txt", olTXT
        End If
End Sub
```

This Visual Basic for Applications example shows you how to create a template using the **Save As** method.

```
Sub CreateTemplate()
    Dim myOlApp As New Outlook.Application
    Dim MyItem As Outlook.MailItem
    Set myOlApp = CreateObject("Outlook.Application")
    Set MyItem = myOlApp.CreateItem(olMailItem)
    MyItem.Subject = "Status Report"
    MyItem.To = "Dan Wilson"
    MyItem.Display
    MyItem.SaveAs "C:\statusrep.oft", OlSaveAsType.olTemplate
End Sub
```



### SaveAsFile Method

Saves the attachment to the specified path.

#### expression.SaveAsFile(Path)

*expression* Required. An expression that returns an <u>Attachment</u> object.

*Path* Required **String**. The location at which to save the attachment.

This Visual Basic for Applications (VBA) example uses the <u>SaveAsFile</u> method to save the first attachment of the currently open <u>item</u> as a file in the C:\ folder, using the attachment's display name as the file name.

```
Sub SaveAttachment()
        Dim myOlApp As Outlook.Application
        Dim myInspector As Outlook. Inspector
        Dim myItem As Outlook.MailItem
        Dim myAttachments As Outlook.Attachments
        Set myOlApp = CreateObject("Outlook.Application")
        Set myInspector = myOlApp.ActiveInspector
        If Not TypeName(myInspector) = "Nothing" Then
                If TypeName(myInspector.CurrentItem) = "MailItem" Th
                        Set myItem = myInspector.CurrentItem
                        Set myAttachments = myItem.Attachments
                        'Prompt the user for confirmation
                        Dim strPrompt As String
                        strPrompt = "Are you sure you want to save t
                        If MsgBox(strPrompt, vbYesNo + vbQuestion) =
                                myAttachments.Item(1).SaveAsFile "C:
                                myAttachments.Item(1).DisplayName
                        End If
                Else
                        MsgBox "The item is of the wrong type."
                End If
        End If
```

```
End Sub
```

If you use Microsoft Visual Basic Scripting Edition (VBScript) in a Microsoft Outlook form, you do not create the <u>Application</u> object. This example shows how to use VBScript code to save the first attachment in the current item.

Else MsgBox "The item is of the wrong type." End If End Sub

# **SelectFolder Method**

Displays the folder if it can be displayed simultaneously with the current folder in the explorer. Otherwise, the explorer switches to the specified folder.

expression.SelectFolder(MAPIFolder)

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

*MAPIFolder* Required. The **MAPIFolder** object representing the folder to be displayed in the explorer.

### Remarks

You can display multiple calendars simultaneously using this method.

This Microsoft Visual Basic for Applications (VBA) example displays a shared calendar simultaneously with the current user's default **Calendar** folder. To run this example without errors, replace Dan Wilson with a valid recipient name whose **Calendar** is shared and whose calendar you have permissions to view.

```
Sub DispCalendars()
    Dim myOlApp As Outlook.Application
    Dim myNms As Outlook.NameSpace
    Dim myFolder As Outlook.MAPIFolder
    Dim myRecipient As Outlook.Recipient
    Dim myExplorer As Outlook.Explorer
    Dim SharedFolder As Outlook.MAPIFolder
    Set myOlApp = CreateObject("Outlook.Application")
    Set myNms = myOlApp.GetNamespace("MAPI")
    Set myFolder = myNms.GetDefaultFolder(olFolderCalendar)
    Set myExplorer = myOlApp.ActiveExplorer
    Set myExplorer.CurrentFolder = myFolder
    Set myRecipient = myNms.CreateRecipient("Dan Wilson")
    Set SharedFolder = myNms.GetSharedDefaultFolder(myRecipient, olF
    myExplorer.SelectFolder SharedFolder
End Sub
```

# Send Method

Sends the appointment, meeting item, e-mail message, or task.

expression.Send

*expression* Required. An expression that returns an <u>AppointmentItem</u>, <u>MeetingItem</u>, <u>MailItem</u>, or <u>TaskItem</u> object.

### Remarks

When you run a program that uses the Microsoft Outlook object model to call the **Send** method, you receive a warning message. This warning message tells you that a program is trying to send a message on your behalf and asks if you want to allow the message to be sent. The warning message contains both a **Yes** and a **No** button. However, the **Yes** button is not available until five seconds have passed since the warning message appeared. You can dismiss the warning message immediately if you click **No**.

This Visual Basic for Applications (VBA) example uses <u>CreateItem</u> to create a simple task and delegate it as a task request to another user. Replace 'Dan Wilson' with a valid recipient name before running this example.

End Sub

If you use Microsoft Visual Basic Scripting Edition (VBScript) in an Outlook form, you do not create the **Application** object, and you cannot use named constants. This example shows how to forward a mail item using VBScript code.

```
Sub CommandButton1_Click()
Set myNameSpace = Application.GetNameSpace("MAPI")
Set myFolder = myNameSpace.GetDefaultFolder(6)
Set myForward = myFolder.Items(1).Forward
myForward.Recipients.Add "Laura Jennings"
myForward.Send
End Sub
```

# **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.

#### 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	MemberCount
Children	RecurrenceState
Class	ReplyRecipients
Companies	ResponseState
Contacts	Saved
DLName	Sent
EntryID	Submitted

HTMLBody	VotingOptions
ReceivedOnBehalfOfEntryID	BodyFormat
ReceivedByEntryID	IsConflict
DownloadState	InternetCodePage
MeetingWorkspaceURL	AutoResolvedWinner

**Note** The **ConversationIndex** property cannot be cached using the **SetColumns** method in Office Outlook 2003. However, this property will not result in an error like the other properties listed above.

The following Visual Basic for Applications (VBA) example uses the **Items** collection to get the items in default Tasks folder, caches the Subject and DueDate properties and then displays the subject and due dates each in turn.

```
Sub SortByDueDate()
        Dim myOlApp As New Outlook.Application
        Dim myNameSpace As Outlook.NameSpace
        Dim myFolder As Outlook.MAPIFolder
        Dim myItem As Object
        Dim myItems As Outlook. Items
        Set myNameSpace = myOlApp.GetNamespace("MAPI")
        Set myFolder = myNameSpace.GetDefaultFolder(olFolderTasks)
        Set myItems = myFolder.Items
        myItems.SetColumns ("Subject, DueDate")
        For Each myItem In myItems
                MsgBox myItem.Subject & " " & myItem.DueDate
        Next myItem
```

End Sub

# SetControlItemProperty Method

Binds an Outlook object model property to a control on an inspector.

#### expression.SetControlItemProperty(Control, PropertyName)

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

*Control* Required **Object**. The control that will be bound to a property.

*PropertyName* Required **String**. The name of the property that will be bound to the control.

### Remarks

You can also use the following line of code

myPage.Controls("bar").ItemProperty = "subject" to bind the subject
property to a control. However, note that this will trigger the security warning if
the property is protected by the object model security guard such as **To**. You can
use the **SetControlItemProperty** method to avoid security warnings with
trusted objects.

The following Visual Basic for Applications (VBA) code adds a custom page to an appointment item, adds a custom textbox control, and binds that control to Subject property.

```
Sub Example()
    Dim myIns As Outlook. Inspector
    Dim myAppt As Outlook.AppointmentItem
    Dim ctrl As Object
    Dim ctrls As Object
    Dim myPages As Outlook.Pages
    Dim myPage As Object
    Set myAppt = Application.CreateItem(olAppointmentItem)
    Set myIns = apti.GetInspector
    Set myPages = myIns.ModifiedFormPages
    Set myPage = myPages.Add("New Page")
    myIns.ShowFormPage ("New Page")
    Set ctrls = myPage.Controls
    Set ctrl = ctrls.Add("Forms.TextBox.1")
    myIns.SetControlItemProperty ctrl, "Subject"
    myAppt.Display
End Sub
```



# 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.

This Visual Basic for Applications (VBA) example uses the **SetCurrentFormPage** method to show the All Fields page of the currently open <u>item</u>. If no items are currently open, a message box will inform the user.

```
Sub ShowAllFieldsPage()
Dim myOlApp As New Outlook.Application
Dim myInspector As Inspector
Dim myItem As Object
Set myInspector = myOlApp.ActiveInspector
On Error GoTo ErrorHandler
myInspector.SetCurrentFormPage ("All Fields")
Set myItem = myInspector.CurrentItem
myItem.Display
Exit Sub
ErrorHandler:
    MsgBox "No current item to display."
End Sub
```



### **SetIcon Method**

Sets the icon for the specified shortcut on the **<u>Shortcuts pane</u>**.

#### expression.SetIcon(Icon)

*expression* Required. An expression that returns an <u>OutlookBarShortcut</u> object.

*Icon* Required **Variant.** The path of the icon.

The following Microsoft Visual Basic for Applications (VBA) example creates a group called MicrosoftSites and adds a shortcut to the Microsoft Network Web page. Then it sets the icon of the shortcut to the icon image MSN.ico located on the user's computer. The example assumes that this icon exists in the specified location.

```
Sub CreateMSNShortcutWithIcon()
    Dim outApp As New Outlook.Application
    Dim exp As Outlook.Explorer
    Dim pans As Outlook.Panes
    Dim bpan As Outlook.OutlookBarPane
    Dim bgrps As Outlook.OutlookBarGroups
    Dim bgrp As Outlook.OutlookBarGroup
    Dim bscs As Outlook.OutlookBarShortcuts
    Dim bsc As Outlook.OutlookBarShortcut
    Dim bsc2 As Outlook.OutlookBarShortcut
    Set exp = outApp.ActiveExplorer
    Set pans = exp.Panes
    Set bpan = pans.Item("OutlookBar")
    Set bgrps = bpan.Contents.Groups
    Set bgrp = bgrps.Add("MicrosoftSites")
    Set bscs = bgrp.Shortcuts
    Set bsc = bscs.Add("http://www.msn.com", "MSN Home Page")
    bsc.SetIcon "C:\MSN.ico"
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.

The following Microsoft Visual Basic/Visual Basic for Applications (VBA) 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 ShowCategoriesDialog
   Dim appolApp As Outlook.Application
   Dim olApptItem As Outlook.AppointmentItem
   'Create an instance of the application
   Set appolApp = New 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 = "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.

This Visual Basic for Applications (VBA) example uses the **ShowFormPage** method to show the All Fields page of the currently open <u>item</u>. If there is no currently open item, a message box will inform the user.

```
Sub ShowAllFieldsPage()
Dim myOlApp As New Outlook.Application
Dim myInspector As Outlook.Inspector
Dim myItem As Object
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."
End Sub
```



### **ShowPane Method**

Displays or hides a specific pane in the explorer.

**Note** You can also use the **<u>Visible</u>** property of the **<u>OutlookBarPane</u>** object to display or hide the <u>Shortcuts pane</u>.

#### 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 olNavigationPane

*Visible* Required. **True** to make the pane visible, **False** to hide the pane.

This Microsoft Visual Basic/Visual Basic for Applications (VBA) example uses the **ShowPane** and **IsPaneVisible** methods to hide the preview pane if it is visible or to display it if it is hidden.

```
Sub ShowHidePreviewPane()
    Dim myOlApp As New Outlook.Application
    Dim myOlExp As Outlook.Explorer
    Set myOlExp = myOlApp.ActiveExplorer
    myOlExp.ShowPane olPreview, _
    Not myOlExp.IsPaneVisible(olPreview)
End Sub
```

If you use Microsoft Visual Basic Scripting Edition (VBScript) in a Microsoft Outlook form, you do not create the <u>Application</u> object. This example shows how to perform the same task using VBScript.

```
Sub CommandButton1_Click()
Set myOlExp = Application.ActiveExplorer
myOlExp.ShowPane 3, Not myOlExp.IsPaneVisible(3)
End Sub
```

# **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 **<u>TaskItem</u>** 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 **<u>Reminder</u>** 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.

The following Microsoft Visual Basic/Visual Basic for Applications (VBA) example delays all active reminders by a specified amount of time.

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 <u>AddressEntries</u> object. Optional Variant for the <u>Results</u> 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 **<u>Items</u>** collection, **Sort** cannot be used and will cause an error with the following properties:

Categories	LastFirstSpaceOnly
Children	LastFirstSpaceOnlyCompany
Class	MemberCount
Companies	NetMeetingAlias
CompanyLastFirstNoSpace	NetMeetingAutoStart

#### CompanyLastFirstSpaceOnly NetMeetingOrganizerAlias

Contacts	NetMeetingServer
DLName	NetMeetingType
IsOnlineMeeting	RecurrenceState
LastFirstAndSuffix	ResponseState
LastFirstNoSpace	Sent
LastFirstNoSpaceCompany	Saved

**Sort** only affects the order of items in a collection. It does not affect the order of items in an explorer view.

The following Visual Basic for Applications (VBA) example uses the **Sort** method to sort the **Items** collection for the default **Tasks** folder by the "DueDate" property and displays the due dates each in turn.

```
Sub SortByDueDate()
        Dim myOlApp As New Outlook.Application
        Dim myNameSpace As Outlook.NameSpace
        Dim myFolder As Outlook.MAPIFolder
        Dim myItem As Outlook.TaskItem
        Dim myItems As Outlook. Items
        Set myNameSpace = myOlApp.GetNamespace("MAPI")
        Set myFolder = myNameSpace.GetDefaultFolder(olFolderTasks)
        Set myItems = myFolder.Items
        myItems.Sort "[DueDate]", False
        For Each myItem In myItems
                MsqBox myItem.Subject & "-- " & myItem.DueDate
        Next myItem
```

```
End Sub
```

If you use Microsoft Visual Basic Scripting Edition (VBScript) in a Microsoft Outlook form, you do not create the **Application** object, and you cannot use named constants. This example shows how to sort the **Contacts** folder by CompanyName property using VBScript.

```
Sub CommandButton1_Click()
 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
End Sub
```

# **Start Method**

Begins synchronizing a user's folders using the specified Send\Receive group.

expression.Start

*expression* Required. An expression that returns a <u>SyncObject</u> object.

This Microsoft Visual Basic/Visual Basic for Applications (VBA) example displays all the Send\Receive groups set up for the user and starts the synchronization based on user's response.

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 <u>StatusUpdateRecipients</u> property) with the current status for the task. Returns an **Object** representing the status report.

expression.StatusReport

*expression* Required. An expression that returns a **<u>TaskItem</u>** object.

This Visual Basic for Applications (VBA) example uses the <u>StatusReport</u> method to report the status of the currently open task.

```
Sub SendStatusReport()
        Dim myOlApp As New Outlook.Application
        Dim myTask As Outlook.TaskItem
        Dim myinspector As Outlook. Inspector
        Dim myReport As Object
        Set myinspector = myOlApp.ActiveInspector
        If Not TypeName(myinspector) = "Nothing" Then
                If TypeName(myinspector.CurrentItem) = "TaskItem" Th
                        Set myTask = myinspector.CurrentItem
                        Set myReport = myTask.StatusReport
                        myReport.Send
                Else
                        MsgBox "No task item is currently open."
                End If
        Else
                MsgBox "No inspector is currently open."
        End If
End Sub
```

If you use Microsoft Visual Basic Scripting Edition (VBScript) in a Microsoft Outlook form, you do not create the <u>Application</u> object. This example shows how to perform the same task using VBScript.

```
Sub CommandButton1_Click()
    If TypeName(Item) = "TaskItem" Then
        Set myReport = Item.StatusReport
        myReport.Send
    Else
        MsgBox "The current item is not a task item."
    End If
End Sub
```



## **Stop Method**

#### As it applies to the **SyncObject** object

Immediately ends synchronizing a user's folders using the specified Send\Receive group. This method does not undo any synchronization that has already occurred.

expression.Stop

*expression* Required. An expression that returns the **SyncObject** object.

As it applies to the **Search** object

Immediately ends the search that is being performed currently.

expression.Stop

*expression* Required. An expression that returns the **Search** object.

This Microsoft Visual Basic/Visual Basic for Applications (VBA) example displays all the Send\Receive groups set up for the user and starts the synchronization based on the user's response. The sub routine following the one below immediately stops the synchronization.

The syc variable is declared as a public variable so it can be referenced by both the sub routines.

```
Public syc As Outlook.SyncObject
Public Sub Sync()
        Dim nsp As Outlook.NameSpace
        Dim sycs As Outlook.SyncObjects
        Dim i As Integer
        Dim strPrompt As Integer
        Set nsp = Application.GetNamespace("MAPI")
        Set sycs = nsp.SyncObjects
        For i = 1 To sycs.Count
                Set syc = sycs.Item(i)
                strPrompt = MsgBox("Do you wish to synchronize " & s
                If strPrompt = vbYes Then
                        syc.Start
                End If
        Next
End Sub
Private Sub StopSync()
    MsgBox "Synchronization stopped by the user."
    syc.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 <u>JournalItem</u> object.

# **Update Method**

The **Update** method posts a change to the **<u>AddressEntry</u>** 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**.

### **Account Property**

Returns or sets a **String** representing the account for the contact. Read/write.

#### expression.Account

*expression* Required. An expression that returns a <u>ContactItem</u> object.



# **Actions Property**

Returns an <u>Actions</u> collection that represents all the available actions for the Outlook <u>item</u>.

#### expression.Actions

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

This Visual Basic for Applications (VBA) example creates a new mail item and uses the **Add** method to add an **Action** to it. Then it sends the mail item to the current user. The mail item received will have the '**Agree**' action in addition to the standard actions such as 'Reply' and 'Reply All'.

```
Sub AddAction()
        Dim myolApp As New Outlook.Application
        Dim myItem As Outlook.MailItem
        Dim myAction As Outlook.Action
        Set myItem = myOlApp.CreateItem(olMailItem)
        Set myAction = myItem.Actions.Add
        myAction.Name = "Agree"
        myItem.To = myolApp.GetNamespace("MAPI").CurrentUser
        myItem.Send
```

End Sub

The following Visual Basic for Applications example creates a new mail item and uses the **Add** method to add an **Action** called 'Link Original' to it. Executing this action will insert a link to the original mail item.

```
Sub AddAction2()
 Dim myOlApp As New Outlook.Application
Dim myItem As Outlook.MailItem
Dim myAction As Outlook.Action
 Set myItem = myOlApp.CreateItem(olMailItem)
Set myAction = myItem.Actions.Add
myAction.Name = "Link Original"
myAction.ShowOn = olMenuAndToolbar
myAction.ReplyStyle = olLinkOriginalItem
myItem.To = "Dan Wilson"
myItem.Send
End Sub
```

If you use Microsoft Visual Basic Scripting Edition (VBScript) in a Microsoft Outlook form, 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
myAction.Name = "Link Original"
myAction.ShowOn = 2
myAction.ReplyStyle = 4
myItem.To = "Dan Wilson"
myItem.Send
```

# **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 <u>AddressEntry</u> object; read-only for the <u>Recipient</u> object.

**Note** The **Address** property must be set before calling the **<u>Details</u>** method.

#### expression.Address

*expression* 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 **MAPIFolder** object representing a contact items folder. Read/write.

#### expression.AddressBookName

*expression* Required. An expression that returns a **MAPIFolder** object representing a contact items folder.

### Remarks

If you try to set the **AddressBookName** property in a non-Contacts items folder, an error will be returned.

The following example changes the Address Book name for the contact items folder and displays the new name to the user. The subroutine accepts the folder object and a **String** representing the new address book name.

Sub BookName()

Dim olApp As Outlook.Application Dim nmsName As Outlook.NameSpace Dim fldFolder As Outlook.MAPIFolder Dim strAns As String Set olApp = New Outlook.Application 'Create a reference to namepsace Set nmsName = olApp.GetNamespace("MAPI") 'Create an instance of the Contacts folder Set fldFolder = nmsName.GetDefaultFolder(olFolderContacts) 'Prompt user for input strAns = InputBox("Type the name of the new address book") 'Call Sub procedure Call Changebook(fldFolder, strAns)

End Sub

Sub Changebook(ByRef fldFolder As MAPIFolder, ByVal strName As Strin 'Changes the name of the address book for a given folder

End Sub

## **AddressEntries Property**

Returns the **<u>AddressEntries</u>** collection for the specified object.

expression.AddressEntries

*expression* Required. An expression that returns an <u>AddressList</u> 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 **<u>Recipient</u>** object.



## **AddressLists Property**

Returns an **AddressLists** collection representing a collection of the address lists available for this <u>session</u>. 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 **<u>NameSpace</u>** object.

## **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 <u>ContactItem</u> object.

## **AnswerWizard Property**

Returns the **AnswerWizard** object for the application.

expression.AnswerWizard()

*expression* Required. An expression that returns an <u>Application</u> 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 <u>SyncObjects</u> object.

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 <u>Application</u> object that represents the parent application (Microsoft Outlook) for an object. Read-only.

### expression.Application

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

This Visual Basic for Applications (VBA) example uses the **Application** property to access Outlook and then creates a new **MailItem** and displays the version of Outlook used to create the item.

```
Sub CreateMailItem()
   Dim myolApp As Outlook.Application
   Dim myItem As Outlook.MailItem
   Set myolApp = CreateObject("Outlook.Application")
   Set myItem = myolApp.CreateItem(olMailItem)
   MsgBox myItem.Application.Version
End Sub
```

If you use VBScript, you do not use the **Application** property to retrieve the **Application** object. Instead, you reference the **Application** object directly.

```
Set myItem = Application.CreateItem(0)
myItem.Display
```

# **AppointmentItem Property**

Returns the **<u>AppointmentItem</u>** object that is the exception. Not valid for deleted appointments.

### expression.AppointmentItem

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

This Visual Basic for Applications (VBA) 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 Microsoft Visual Basic Scripting Edition (VBScript), see the Note at the end of the example.

```
Public Sub cmdExample()
    Dim myOlApp As Outlook.Application
    Dim myApptItem As Outlook.AppointmentItem
    Dim myRecurrPatt As Outlook.RecurrencePattern
    Dim myNamespace As Outlook.NameSpace
    Dim myFolder As Outlook.MAPIFolder
    Dim myItems As Outlook. Items
    Dim myDate As Date
    Dim myOddApptItem As Outlook.AppointmentItem
    Dim saveSubject As String
    Dim newDate As Date
    Dim myException As Outlook.Exception
    Set myOlApp = New Outlook.Application
    Set myApptItem = myOlApp.CreateItem(olAppointmentItem)
   myApptItem.Start = #2/2/2003 3:00:00 PM#
   myApptItem.End = #2/2/2003 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 2/2/03 and ends on 2/2/04
```

```
'and save it.
    Set myRecurrPatt = myApptItem.GetRecurrencePattern
   myRecurrPatt.RecurrenceType = olRecursDaily
   myRecurrPatt.PatternStartDate = #2/2/2003#
    myRecurrPatt.PatternEndDate = #2/2/2004#
    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 3/12/03.
    myDate = #3/12/2003 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 = #3/12/2003 3:30:00 PM#
    mvOddApptItem.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.
    MsqBox 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, a few changes 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, therefore:

```
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

## **AssistantName Property**

Returns or sets a **String** representing the name of the person who is the assistant for the contact. Read/write.

#### expression.AssistantName

*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 **<u>ContactItem</u>** object.



## **Attachments Property**

Returns an <u>Attachments</u> object that represents all the attachments for the specified <u>item</u>.

### expression.Attachments

This Visual Basic for Applications (VBA) example uses the **<u>Remove</u>** method to remove all attachments from a forwarded message before sending it on to 'Dan Wilson'. To run this example, replace 'Dan Wilson' with a valid recipient's name and keep an item with attachments open in an inspector window.

```
Sub RemoveAttachmentBeforeForwarding()
    Dim myolApp As Outlook.Application
    Dim myinspector As Outlook. Inspector
    Dim myItem As Outlook.MailItem
    Dim myattachments As Outlook.Attachments
    Set myolApp = CreateObject("Outlook.Application")
    Set myinspector = myolApp.ActiveInspector
    If Not TypeName(myinspector) = "Nothing" Then
        Set myItem = myinspector.CurrentItem.Forward
        Set myattachments = myItem.Attachments
        While myattachments.Count > 0
               myattachments.Remove 1
        Wend
        myItem.Display
        myItem.Recipients.Add "Dan Wilson"
        myItem.Send
    Else
        MsgBox "There is no active inspector."
    End If
End Sub
```

## **AutoForwarded Property**

**True** if the mail message was automatically forwarded. Read/write **Boolean**.

#### expression.AutoForwarded

## **AutoResolvedWinner Property**

Returns a **Boolean** that determines if the item is a winner of an automatic conflict resolution. Read-only.

**Note** A value of **False** does not necessarily indicate that the item is a loser of an automatic conflict resolution. The item should be in conflict with another item.

#### expression.AutoResolvedWinner

## Remarks

If an item has its **Conflicts.Count** property greater than zero and if its **AutoResolvedWinner** property is **True**, it is a winner of an automatic conflict resolution. On the other hand, if the item is in conflict and has its **AutoResolvedWinner** property as **False**, it is a loser in an automatic conflict resolution.

The following Microsoft Visual Basic for Applications (VBA) example used the **AutoResolvedWinner** property to determine if an item is a winner or loser in an automatic conflict resolution. To run this example, make sure an e-mail item is open in the active window.

```
Sub ConflictStatus()
Dim myOlApp As New Outlook.Application
Dim mail As Outlook.MailItem
Set mail = myOlApp.ActiveInspector.CurrentItem
If mail.Conflicts.Count > 0 Then
If mail.AutoResolvedWinner = True Then
MsgBox "This item is a winner in an automatic conflict resolution
Else
MsgBox "This item is a loser in an automatic conflict resolution.
End If
Else
MsgBox "This item is not in conflict with any item."
End If
End Sub
```

## **AutoResponse Property**

Returns or sets a **String** representing the text of an automatic response for a **<u>Recipient</u>**. Read/write.

expression.AutoResponse

*expression* Required. An expression that returns a **<u>Recipient</u>** 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 <u>item</u>. This is a free-form text field. Read/write.

#### expression.BillingInformation

## **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 Microsoft Outlook <u>item</u>. Read/write.

The **BodyFormat** property allows you to programmatically change the editor that is used for the body of an item.

#### expression.Body

### Remarks

Microsoft Office Outlook 2003 inherits the Outlook 2002 object model guard behavior. In addition, it blocks code that attempts to access the **Body** property of various Outlook items. This allows users to verify that the program or add-in accessing the **Body** property of items is trustworthy, before they allow access to the contents of the items. Even though this leads to the display of security warnings in the existing COM add-ins that access the **Body** property of items, this will help prevent malicious code from running without the user being aware of it.

You can avoid the display of security warnings by deriving all objects, properties, and methods from the **Application** object passed in the **OnConnection** procedure of the add-in. Outlook trusts only the **Application** object passed in the **OnConnection** procedure of the add-in. If you create a new **Application** object— for example, by using the **CreateObject** method— that object and any of its subordinate objects, properties, and methods will not be trusted and the blocked properties and methods will throw security warnings.

This Microsoft Visual Basic Scripting Edition (VBScript) example uses the **Open** event of an <u>item</u> 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 (RTF), and HTML. Read/write.

OlBodyFormat can be one of the following OlBodyFormat constants.

olFormatHTML olFormatPlain olFormatRichText olFormatUnspecified

#### expression.BodyFormat

## Remarks

All text formatting will be lost when the **BodyFormat** property is switched from RTF to HTML and vice-versa.

In earlier versions of Outlook, the **BodyFormat** property returned the **olFormatUnspecified** constant for a newly created item that has not been displayed or whose **BodyFormat** property is not yet set programmatically. In Microsoft Office Outlook 2003, the property returns the format that is currently set in the Outlook user interface.

The following Microsoft Visual Basic/Visual Basic for Applications (VBA) example creates a new **MailItem** object and sets the **BodyFormat** property to **olFormatHTML**. The body text of the e-mail item will now appear in HTML format.

```
Sub CreateHTMLMail()
'Creates a new e-mail item 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
'Set body format to HTML
.BodyFormat = olFormatHTML
.HTMLBody = "<HTML><H2>The body of this message will appear i
.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

# **BusinessAddress Property**

Returns or sets a **String** representing the whole, unparsed business address for the contact. Read/write.

#### expression.BusinessAddress

# **BusinessAddressCity Property**

Returns or sets a **String** representing the city name portion of the business address for the contact. Read/write.

#### expression.BusinessAddressCity

### 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

### 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

### 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

### 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

### 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

### 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

# **BusinessHomePage Property**

Returns or sets a **String** representing the URL of the business Web page for the contact. Read/write.

#### expression.BusinessHomePage

# **BusinessTelephoneNumber Property**

Returns or sets a **String** representing the first business telephone number for the contact. Read/write.

expression.BusinessTelephoneNumber



### **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

# CallbackTelephoneNumber Property

Returns or sets a **String** representing the callback telephone number for the contact. Read/write.

#### expression.CallbackTelephoneNumber

# **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

# **CarTelephoneNumber Property**

Returns or sets a **String** representing the car telephone number for the contact. Read/write.

#### expression.CarTelephoneNumber



## **Categories Property**

Returns or sets a **String** representing the categories assigned to the Microsoft Outlook <u>item</u>. Read/write.

expression.Categories

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

### Example

The following Microsoft Visual Basic/Visual Basic for Applications (VBA) example creates a new appointment, displays the appointment on the screen, and opens the **Show Categories** dialog box. Finally, it displays the categories that the user assigned using **ShowCategoriesDialog**. Replace 'Dan Wilson' with a valid recipient name before running the example.

```
Sub Appointment()
'Creates an appointment to access ShowCategoriesDialog
    Dim appolApp As Outlook.Application
    Dim olApptItem As Outlook.AppointmentItem
    'Creates an instance of the application
    Set appolApp = Outlook.Application
    'Creates appointment item
    Set olApptItem = appolApp.CreateItem(olAppointmentItem)
    olApptItem.Body = "Please meet with me regarding these sales fig
    olApptItem.Recipients.Add ("Dan Wilson")
    olApptItem.Subject = "Sales Reports"
    'Display the appointment
    olApptItem.Display
    'Display the Show Categories dialog box
    olApptItem.ShowCategoriesDialog
    MsgBox olApptItem.Categories
End Sub
```

# **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

*expression* 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



## **Class Property**

Returns an **<u>OlObjectClass</u>** 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 olConflict olConflicts 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

olPost olPropertyPages olPropertyPageSite olRecipient olRecipients olRecurrencePattern olRemote olReport olResults olSearch olSelection olSyncObject olSyncObjects olTableViewField olTableViewFields olTask olTaskRequest olTaskRequestAccept olTaskRequestDecline olTaskRequestUpdate olTimelineViewField olTimelineViewFields olUserProperties olUserProperty olViews

expression.Class

*expression* Required. An expression that returns a Microsoft 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.

```
Sub CreatePersonalNote()
    Dim myOlApp As Outlook.Application
    Dim myItem As Outlook.NoteItem
    Set myOlApp = CreateObject("Outlook.Application")
    Set myItem = myOlApp.CreateItem(olNoteItem)
    myItem.Color = olBlue
    myItem.Display
End Sub
```

If you use VBScript, you do not create the <u>Application</u> object, and you cannot use named constants. This example shows how to use the **Color** property using 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

*expression* Required. An expression that returns an <u>Application</u> object.

### Example

This Microsoft Visual Basic for Applications (VBA) example displays the number of COM add-ins currently loaded.

```
Dim myOlApp As New Outlook.Application
Private Sub CountCOMAddins()
    MsgBox "There are " & _
    myOlApp.COMAddIns.Count & " COM add-ins."
End Sub
```

If you use Microsoft Visual Basic Scripting Edition (VBScript) in a Microsoft Outlook form, you do not declare an <u>Application</u> object variable. This example shows how to perform the same task using VBScript code in an Outlook form.

```
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**.

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 <u>item</u>. 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 <u>ContactItem</u> 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 <u>CompanyName</u>, <u>LastName</u>, <u>FirstName</u> and <u>MiddleName</u> 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 <u>ContactItem</u> 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 **<u>ContactItem</u>** 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

*expression* Required. An expression that returns a <u>ContactItem</u> object.

## **CompanyName Property**

Returns or sets a **String** representing the company name for the contact. Read/write.

expression.CompanyName

*expression* Required. An expression that returns a <u>ContactItem</u> object.

### **Complete Property**

**True** if the task is completed. Read/write **Boolean**.

expression.Complete

*expression* Required. An expression that returns a **<u>TaskItem</u>** 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 <u>ContactItem</u> object.

# ConferenceServerAllowExternal Property

Reserved for future use.

# **ConferenceServerPassword Property**

Reserved for future use.

## **Conflicts Property**

Return the **Conflicts** object that represents the items that are in conflict for any Microsoft Outlook item object. Read-only.

expression.Conflicts

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

### Example

The following Microsoft Visual Basic for Applications (VBA) example uses the **Count** property of the **Conflicts** object to determine if the item is involved in any conflict. To run this example, make sure a mail item is open in the active window.

### **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**

Sets or returns a **String** representing the contact names associated with the task item or journal entry. Read/write.

### expression.ContactNames

*expression* Required. An expression that returns a **JournalItem** or **TaskItem** object.

### Remarks

If a program tries to reference any type of recipient information by using the Outlook object model, a dialog box is displayed that asks you to confirm access to this information. You can allow access to the Address Book or recipient information for up to ten minutes after you receive the dialog box. This allows features, such as mobile device synchronization, to be completed.

You receive the confirmation dialog box when a solution tries to programmatically access the **ContactNames** property.



### **Contents Property**

Returns the **<u>OutlookBarStorage</u>** object for the specified <u>Outlook Bar</u> 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 <u>Application</u> 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
MsgBox myMsg
```



## **ConversationIndex Property**

Returns a **String** representing the index of the conversation thread of the <u>item</u>. 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 <u>item</u>. 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 <u>item</u> 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 <u>Action</u> object.

## **Count Property**

Returns a **Long** indicating the count of objects in the specified collection. Read-only.

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 <u>item</u>. This property corresponds to the MAPI property PR\_CREATION\_TIME. Read-only.

### expression.CreationTime

*expression* 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 (VBA) example uses the <u>CurrentFolder</u> property to change the displayed folder to the user's **Calendar** folder.

```
Sub ChangeCurrentFolder()
    Dim myolApp As Outlook.Application
    Dim myNamespace As Outlook.NameSpace
    Set myolApp = CreateObject("Outlook.Application")
    Set myNamespace = myolApp.GetNamespace("MAPI")
    Set myolApp.ActiveExplorer.CurrentFolder = ______
    myNamespace.GetDefaultFolder(olFolderCalendar)
End Sub
```

End Sub

If you use Microsoft Visual Basic Scripting Edition (VBScript) in a Microsoft Outlook form, you do not create the <u>Application</u> object, and you cannot use named constants. This example shows how to perform the same task using VBScript code.

```
Set myNameSpace = Application.GetNameSpace("MAPI")
Set Application.ActiveExplorer.CurrentFolder = _
    myNameSpace.GetDefaultFolder(9)
```



# **CurrentGroup Property**

With the new <u>Navigation Pane</u> and <u>Shortcuts pane</u> in Microsoft Office Outlook 2003, this property does not have any use in Office Outlook 2003.



### **CurrentItem Property**

Returns an **Object** representing the current <u>item</u> being displayed in the inspector.

**Note** If no item is currently open, an error message will be returned.

#### expression.CurrentItem

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

### Example

This Visual Basic for Applications (VBA) example uses the <u>CurrentItem</u> property to obtain the current <u>item</u> that the user is viewing and closes it. If no item is currently open, an error message will be returned.

```
Sub CloseItem()
        Dim myOlApp As New Outlook.Application
        Dim myItem As Object
        Set myItem = myOlApp.ActiveInspector.CurrentItem
        myItem.Close olSave
End Sub
```

## **CurrentUser Property**

Returns the display name of the currently logged-on user as a **<u>Recipient</u>** object. Read-only.

#### expression.CurrentUser

*expression* Required. An expression that returns a **<u>NameSpace</u>** object.

### Remarks

Outlook blocks code that attempts to access the **CurrentUser** property for security reasons. If you run a third-party add-in, custom solution, or other program that uses the **CurrentUser** property in Office Outlook 2003, you may receive the following warning:

A program is trying to access e-mail addresses you have stored in Outlook. Do you want to allow this? If this is unexpected, it may be a virus and you should choose "No".

### Example

This Visual Basic for Applications (VBA) example uses the <u>CurrentUser</u> property to obtain the name of the currently logged-on user and then displays a message box containing the name.

```
Sub DisplayCurrentUser()
    Dim myolApp As Outlook.Application
    Dim myNamespace As Outlook.NameSpace
    Set myOlApp = CreateObject("Outlook.Application")
    Set myNameSpace = myOlApp.GetNameSpace("MAPI")
    MsgBox myNameSpace.CurrentUser
End Cub
```

End Sub

If you use Microsoft Visual Basic Scripting Edition (VBScript) in a Microsoft Outlook form, you do not create the <u>Application</u> object. This example shows how to perform the same task using VBScript code.

```
Sub CommandButton1_Click()
  Set myNameSpace = Application.GetNameSpace("MAPI")
  MsgBox myNameSpace.CurrentUser
End Sub
```



## **CurrentView Property**

Returns or sets a **View** object (for the **MAPIFolder** object) or **Variant** (for the **Explorer** object) representing the current <u>view</u>. Read-only for the **MAPIFolder** object. Read/write for the **Explorer** object.

#### 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 and **ViewSwitch** takes place after the change is effective.

### Example

The following Visual Basic for Applications (VBA) example sets the current view in the active explorer to messages if the **Inbox** is displayed.

```
Sub ChangeCurrentView()
    Dim myOlApp As New Outlook.Application
    Dim myOlExp As Outlook.Explorer
    Set myOlExp = myOlApp.ActiveExplorer
    If myOlExp.CurrentFolder = "Inbox" Then
        myOlExp.CurrentView = "Messages"
    End If
End Sub
```

If you use Microsoft Visual Basic Scripting Edition (VBScript) in a Microsoft Outlook form, you do not create the <u>Application</u> object. This example shows how to perform the same task using VBScript code.

```
Sub CommandButton1_Click()
   Application.ActiveExplorer.CurrentView = "Messages"
End Sub
```

The following VBA example displays the current view of the Inbox folder.

```
Sub TestMAPIFolderCurrentView()
   Dim nsp As Outlook.NameSpace
   Dim mpFolder As Outlook.MAPIFolder
   Dim vw As Outlook.View
   Dim strView As String
   Set nsp = Application.Session
   Set mpFolder = nsp.GetDefaultFolder(olFolderInbox)
   Set vw = mpFolder.CurrentView
   MsgBox "The Current View is: " & vw.Name
```

End Sub

## **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 <u>ContactItem</u> object.



# **CustomViewsOnly Property**

Returns or sets a **Boolean** that determines which views are displayed on the **View** 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.

### Remarks

This property has an effect only on the **View** menu. It does not affect the display of views in the <u>Navigation Pane</u>.

### 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 can be seen in the user interface.

```
Sub SetCusView()
'Sets the CustomViewsOnly property depending on the user's response
    Dim olApp As Outlook.Application
    Dim nmsName As Outlook.NameSpace
    Dim fldFolder As Outlook.MAPIFolder
    Dim lngAns As Long
    Set olApp = New 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
    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 View
    If lngAns = vbYes Then
        fldFolder.CustomViewsOnly = True
    Else
        fldFolder.CustomViewsOnly = False
    End If
    'Display only custom views
    If lngAns = vbYes Then
       MsgBox "The View menu for the " & fldFolder.Name & " folder w
    'Display all views
    Else
       MsgBox "The View menu for the " & fldFolder.Name & " folder w
    End If
```

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

*expression* Required. An expression that returns a **<u>RecurrencePattern</u>** 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 **<u>RecurrencePattern</u>** 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."

```
Sub CreateAppointment()
    Dim myOlApp As Outlook.Application
    Dim myApptItem As AppointmentItem
    Dim myRecurrPatt As RecurrencePattern
    Set myOlApp = New outlook.Application
    Set myApptItem = myOlApp.CreateItem(olAppointmentItem)
    Set myRecurrPatt = myApptItem.GetRecurrencePattern
    myRecurrPatt.RecurrenceType = olRecursWeekly
    myRecurrPatt.PatternStartDate = #1/21/1998 2:00:00 PM#
    myRecurrPatt.Interval = 3
    myRecurrPatt.PatternEndDate = #12/21/2001 5:00:00 PM#
    myApptItem.Subject = "Important Appointment"
    myApptItem.Save
    myApptItem.Display
```

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.

```
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 **OlltemType** constant indicating the default Outlook <u>item</u> type contained in the folder. Read/write for the **Results** object; read-only for the **MAPIFolder** object.

OlltemType can be one of these OlltemType constants. olAppointmentItem olContactItem olDistributionListItem olJournalItem olMailItem olNoteItem olPostItem

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 **<u>OlTaskDelegationState</u>** 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 **<u>TaskItem</u>** 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**.

#### expression.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.

This Visual Basic/Visual Basic for Applications (VBA) example returns the value of the **<u>Dirty</u>** property as the value of a global variable.

# **DisplayName Property**

For the **<u>Attachment</u>** object:

Returns or sets a **String** representing the name, which does not need to 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 the **FormDescription** object:

Returns or sets a **String** representing the name of the form, which is displayed in the **Choose Forms** dialog box. If both the **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.

This Visual Basic for Applications (VBA) example uses the **SaveAsFile** method to save the first attachment of the currently open item as a file in the C:\ folder, using the attachment's display name as the file name.

```
Sub SaveAttachment()
        Dim myOlApp As Outlook.Application
        Dim myInspector As Outlook. Inspector
        Dim myItem As Outlook.MailItem
        Dim myAttachments As Outlook.Attachments
        Set myOlApp = CreateObject("Outlook.Application")
        Set myInspector = myOlApp.ActiveInspector
        If Not TypeName(myInspector) = "Nothing" Then
                If TypeName(myInspector.CurrentItem) = "MailItem" Th
                        Set myItem = myInspector.CurrentItem
                        Set myAttachments = myItem.Attachments
                        'Prompt the user for confirmation
                        Dim strPrompt As String
                        strPrompt = "Are you sure you want to save t
                        If MsgBox(strPrompt, vbYesNo + vbQuestion) =
                                myAttachments.Item(1).SaveAsFile "C:
                                myAttachments.Item(1).DisplayName
                        End If
                Else
                        MsgBox "The item is of the wrong type."
                End If
        End If
```

```
End Sub
```

If you use Microsoft Visual Basic Scripting Edition (VBScript) in a Microsoft Outlook form, you do not create the **Application** object. This example shows how to perform the same task using VBScript code.

```
Set myAttachments = Item.attachments
                'Prompt the user for confirmation
                Dim strPrompt
                strPrompt = "Are you sure you want to save the first
                If MsgBox(strPrompt, vbYesNo + vbQuestion) = vbYes T
                        myAttachments.Item(1).SaveAsFile "C:\" &
                        myAttachments.Item(1).DisplayName
                End If
```



# **DisplayType Property**

Returns an **<u>OlDisplayType</u>** 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 <u>AddressEntry</u> or <u>Recipient</u> object.

## **DLName Property**

Returns or sets a **String** representing the display name of a distribution list. Read/write.

expression.DLName

This Microsoft Visual Basic/Visual Basic for Applications (VBA) example creates a new distribution list and then prompts the user for a name.

```
Sub CreateDL()
    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 distr
    myDistList.Save
    myDistList.Display
End Sub
```

If you use Microsoft Visual Basic Scripting Edition (VBScript) in a Microsoft Outlook form, you do not create the <u>Application</u> object, and you cannot use named constants. This example shows how to perform the same task using VBScript code.

```
Set myDistList = Application.CreateItem(7)
myDistList.DLName = _
    InputBox("Type the name of the new distribution list.")
myDistList.Save
myDistList.Display
```



## **DocPosted Property**

**True** if the journalized <u>item</u> was posted as part of the journalized <u>session</u>. Read/write **Boolean**.

expression.DocPosted



# **DocPrinted Property**

**True** if the journalized <u>item</u> was printed as part of the journalized <u>session</u>. Read/write **Boolean**.

expression.DocPrinted



# **DocRouted Property**

**True** if the journalized <u>item</u> was routed as part of the journalized <u>session</u>. Read/write **Boolean**.

expression.DocRouted



# **DocSaved Property**

**True** if the journalized <u>item</u> was saved as part of the journalized <u>session</u>. Read/write **Boolean**.

expression.DocSaved



## **DownloadState Property**

Returns an **OlDownloadState** constant indicating the download state of the item. Read-only <u>**OlDownloadState**</u>.

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.

The following Microsoft Visual Basic/Visual Basic for Applications (VBA) example searches through the user's **Inbox** for items that have not yet been fully downloaded. If any not yet fully downloaded items are found, a message is displayed to the user, and the item is marked for download.

```
Sub DownloadItems()
    Dim outApp As Outlook.Application
    Dim mpfInbox As Outlook.MAPIFolder
    Dim objItems As Outlook.Items
    Dim obj As Object
    Dim i As Integer
    Dim iCount As Integer
    Set outApp = CreateObject("Outlook.Application")
    Set mpfInbox = outApp.GetNamespace("MAPI").GetDefaultFolder(olFo
    Set objItems = mpfInbox.Items
    iCount = objItems.Count
    'Loop all items in the Inbox folder
    For i = 1 To iCount
        Set obj = objItems.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
            obj.Save
        End If
    Next
```

End Sub

## **DueDate Property**

Returns or sets a **Date** indicating the due date for the task. Read/write.

### expression.DueDate

## **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 <u>AppointmentItem</u>, <u>RecurrencePattern</u>, or <u>JournalItem</u> object.

This Visual Basic for Applications example uses **<u>CreateItem</u>** to create an appointment and uses <u>**MeetingStatus**</u> to set the meeting status to "Meeting" to turn it into a meeting request with both a required and an optional attendee.

```
Sub ScheduleMeeting()
        Dim myOlApp As Outlook.Application
        Dim myItem as AppointmentItem
        Dim myRequiredAttendee As Recipient
        Dim myOptionalAttendee As Recipient
        Dim myResourceAttendee As Recipient
        Set myOlApp = CreateObject("Outlook.Application")
        Set myItem = myOlApp.CreateItem(olAppointmentItem)
        myItem.MeetingStatus = olMeeting
        myItem.Subject = "Strategy Meeting"
        myItem.Location = "Conference Room B"
        myItem.Start = #9/24/2002 1:30:00 PM#
        myItem.Duration = 90
        Set myRequiredAttendee = myItem.Recipients.Add ("Nate Sun")
        myRequiredAttendee.Type = olRequired
        Set myOptionalAttendee = myItem.Recipients.Add ("Kevin Kenne
        myOptionalAttendee.Type = olOptional
        Set myResourceAttendee = myItem.Recipients.Add("Conference R
        myResourceAttendee.Type = olResource
        myItem.Send
```

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.

```
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



# **EditorType Property**

Returns an **<u>OlEditorType</u>** constant indicating the type of editor. Read-only.

OlEditorType can be one of these OlEditorType constants. olEditorHTML olEditorRTF olEditorText olEditorWord

expression.EditorType

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

This Microsoft Visual Basic Scripting Edition (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 <u>ScriptText</u> 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.</BOD
    '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
    '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.E
    'Access the items's
    'FormDescription object.
    Set myForm = Item.FormDescription
    'Display all the code
    'in the Script Editor.
    MsqBox 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

This Visual Basic for Applications (VBA) example sets "someone@example.com" as the e-mail address for the first e-mail entry of a contact.

```
Sub CreatePeerContact()
   Dim myOlApp As Outlook.Application
   Dim myItem As Outlook.ContactItem
   Set myOlApp = CreateObject("Outlook.Application")
   Set myItem = myOlApp.CreateItem(olContactItem)
   myItem.Email1Address = "someone@example.com"
   myItem.Display
End Sub
```

If you use Microsoft Visual Basic Scripting Edition (VBScript) in a Microsoft Outlook form, you do not create the <u>Application</u> object, and you cannot use named constants. This example shows how to perform the same task using VBScript code.

```
Set myItem = Application.CreateItem(2)
myItem.Email1Address = "someone@example.com"
myItem.Display
```

# **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 e-mail transport. Read/write.

#### expression.Email1AddressType

This Visual Basic for Applications (VBA) example sets "SMTP" as the address type for the first e-mail entry of a contact.

```
Sub SetType()
        Dim myOlApp As New Outlook.Application
        Dim myItem As Outlook.ContactItem
        Set myOlApp = CreateObject("Outlook.Application")
        Set myItem = myOlApp.CreateItem(olContactItem)
        myItem.Email1Address = "someone@example.com"
        myItem.Email1AddressType = "SMTP"
        myItem.Display
```

End Sub

If you use Microsoft Visual Basic Scripting Edition (VBScript) in a Microsoft Outlook form, you do not create the **Application** object, and you cannot use named constants. This example shows how to perform the same task using VBScript code.

```
Set myItem = Application.CreateItem(2)
myItem.Email1Address = "someone@example.com"
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

# **Email1EntryID Property**

Returns a **String** representing the entry ID of the first e-mail address for the contact. Read-only.

### expression.Email1EntryID

# **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

This Visual Basic for Applications (VBA) example sets "someone@example.com" as the e-mail address for the second e-mail entry of a contact.

```
Sub CreatePeerContact()
    Dim myOlApp As Outlook.Application
    Dim myItem As Outlook.ContactItem
    Set myOlApp = CreateObject("Outlook.Application")
    Set myItem = myOlApp.CreateItem(olContactItem)
    myItem.Email2Address = "someone@example.com"
    myItem.Display
End Sub
```

If you use Microsoft Visual Basic Scripting Edition (VBScript) in a Microsoft Outlook form, you do not create the <u>Application</u> object, and you cannot use named constants. This example shows how to perform the same task using VBScript code.

```
Set myItem = Application.CreateItem(2)
myItem.Email2Address = "someone@example.com"
myItem.Display
```

# 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 e-mail transport. Read/write.

#### expression.Email2AddressType

This Visual Basic for Applications (VBA) example sets "SMTP" as the address type for the second e-mail entry of a contact.

```
Sub SetType()
        Dim myOlApp As New Outlook.Application
        Dim myItem As Outlook.ContactItem
        Set myOlApp = CreateObject("Outlook.Application")
        Set myItem = myOlApp.CreateItem(olContactItem)
        myItem.Email2Address = "someone@example.com"
        myItem.Email2AddressType = "SMTP"
        myItem.Display
```

End Sub

If you use Microsoft Visual Basic Scripting Edition (VBScript) in a Microsoft Outlook form, you do not create the **Application** object, and you cannot use named constants. This example shows how to perform the same task using VBScript code.

```
Set myItem = Application.CreateItem(2)
myItem.Email2Address = "someone@example.com"
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

## **Email2EntryID Property**

Returns a **String** representing the entry ID of the second e-mail entry for the contact. Read-only.

#### expression.Email2EntryID

### **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

This Visual Basic for Applications (VBA) example sets "someone@example.com" as the e-mail address for the third e-mail entry of a contact.

```
Sub CreatePeerContact()
   Dim myOlApp As Outlook.Application
   Dim myItem As Outlook.ContactItem
   Set myOlApp = CreateObject("Outlook.Application")
   Set myItem = myOlApp.CreateItem(olContactItem)
   myItem.Email3Address = "someone@example.com"
   myItem.Display
End Sub
```

If you use Microsoft Visual Basic Scripting Edition (VBScript) in a Microsoft Outlook form, you do not create the <u>Application</u> object, and you cannot use named constants. This example shows how to perform the same task using VBScript code.

```
Set myItem = Application.CreateItem(2)
myItem.Email3Address = "someone@example.com"
myItem.Display
```

## 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 e-mail transport. Read/write.

#### expression.Email3AddressType

This Visual Basic for Applications (VBA) example sets "SMTP" as the address type for the third e-mail entry of a contact.

```
Sub SetType()
        Dim myOlApp As New Outlook.Application
        Dim myItem As ContactItem
        Set myOlApp = CreateObject("Outlook.Application")
        Set myItem = myOlApp.CreateItem(olContactItem)
        myItem.Email3Address = "someone@example.com"
        myItem.Email3AddressType = "SMTP"
        myItem.Display
```

End Sub

If you use Microsoft Visual Basic Scripting Edition (VBScript) in a Microsoft Outlook form, you do not create the **Application** object, and you cannot use named constants. This example shows how to perform the same task using VBScript code.

```
Set myItem = Application.CreateItem(2)
myItem.Email3Address = "someone@example.com"
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

## **Email3EntryID Property**

Returns a **String** representing the entry ID of the third e-mail entry for the contact. Read-only.

#### expression.Email3EntryID

### **Enabled Property**

**True** if the action is enabled in the application. Read/write **Boolean**.

#### expression.Enabled

## **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 <u>AppointmentItem</u> or a <u>JournalItem</u> object.

This Visual Basic for Applications (VBA) 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 Microsoft Visual Basic Scripting Edition (VBScript), see the Note at the end of the example.

```
Public Sub cmdExample()
    Dim myOlApp As Outlook.Application
    Dim myApptItem As Outlook.AppointmentItem
    Dim myRecurrPatt As Outlook.RecurrencePattern
    Dim myNamespace As Outlook.NameSpace
    Dim myFolder As Outlook.MAPIFolder
    Dim myItems As Outlook. Items
    Dim myDate As Date
    Dim myOddApptItem As Outlook.AppointmentItem
    Dim saveSubject As String
    Dim newDate As Date
    Dim myException As Outlook.Exception
    Set myOlApp = New Outlook.Application
    Set myApptItem = myOlApp.CreateItem(olAppointmentItem)
   myApptItem.Start = #2/2/2003 3:00:00 PM#
   myApptItem.End = #2/2/2003 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 2/2/03 and ends on 2/2/04

```
'and save it.
    Set myRecurrPatt = myApptItem.GetRecurrencePattern
   myRecurrPatt.RecurrenceType = olRecursDaily
   myRecurrPatt.PatternStartDate = #2/2/2003#
    myRecurrPatt.PatternEndDate = #2/2/2004#
    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 3/12/03.
    myDate = #3/12/2003 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 = #3/12/2003 3:30:00 PM#
    mvOddApptItem.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.
    MsqBox myException.OriginalDate & ": " & saveSubject
    'Display the current date, time, and subject
    'for this exception.
    MsqBox myException.AppointmentItem.Start & ": " &
   myException.AppointmentItem.Subject
End Sub
```

**Note** For this example to work properly in VBScript in a Microsoft Outlook form, a few changes need to be made in the code.

You do not 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 **<u>RecurrencePattern</u>** object.

### **EntryID Property**

Returns a **String** representing the unique entry ID of the object. This property corresponds to the MAPI property **PR\_ENTRYID**. Read-only.

#### expression.EntryID

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

### Remarks

A MAPI store provider assigns a unique ID string when an item is created in its store. Therefore, the **EntryID** property is not set for a Microsoft Outlook item until it is saved or sent. The EntryID changes when an item is moved into another store, for example, from your **Inbox** to a Microsoft Exchange Server public folder, or from one Personal Folders (.pst) file to another .pst file. Solutions should not depend on the **EntryID** property to be unique unless items will not be moved. The **EntryID** property returns a MAPI long-term EntryID. For more information about long- and short-term EntryIDs, search http://msdn.microsoft.com for PR\_ENTRYID.

This Visual Basic for Applications (VBA) example uses the **EntryID** property to compare the entry ID of one contact to the entry ID of a contact returned by a search operation to determine whether the objects represent the same contact. Replace the name with a valid contact name in your Contacts folder before running this example.

```
Sub UseEntryID()
        Dim myOlApp As Outlook.Application
        Dim myNamespace As Outlook.NameSpace
        Dim myContacts As Outlook.MAPIFolder
        Dim myItem1 As Outlook.ContactItem
        Dim myItem2 As Outlook.ContactItem
        Set myOlApp = CreateObject("Outlook.Application")
        Set myNameSpace = myOlApp.GetNamespace("MAPI")
        Set myContacts = myNameSpace.GetDefaultFolder(olFolderContac
        Set myItem1 = myContacts.Items.Find("[FirstName] = ""Dan""")
        Set myitem2 = myContacts.Items.Find("[FileAs] = ""Wil"" and
        If Not TypeName(myitem2) = "Nothing" Then
                If myItem1.EntryID = myitem2.EntryID Then
                        MsgBox "These two contact items refer to the
                End If
        Else
                MsgBox "The contact items were not found."
        End If
End Sub
```

### **Exceptions Property**

Returns the **Exceptions** collection for a specified series of recurring appointments.

expression.Exceptions

*expression* Required. An expression that returns a **<u>RecurrencePattern</u>** object.

This Visual Basic for Applications (VBA) 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 Microsoft Visual Basic Scripting Edition (VBScript), see the Note at the end of the example.

```
Public Sub cmdExample()
    Dim myOlApp As Outlook.Application
    Dim myApptItem As Outlook.AppointmentItem
    Dim myRecurrPatt As Outlook.RecurrencePattern
    Dim myNamespace As Outlook.NameSpace
    Dim myFolder As Outlook.MAPIFolder
    Dim myItems As Outlook. Items
    Dim myDate As Date
    Dim myOddApptItem As Outlook.AppointmentItem
    Dim saveSubject As String
    Dim newDate As Date
    Dim myException As Outlook.Exception
    Set myOlApp = New Outlook.Application
    Set myApptItem = myOlApp.CreateItem(olAppointmentItem)
   myApptItem.Start = #2/2/2003 3:00:00 PM#
   myApptItem.End = #2/2/2003 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 2/2/03 and ends on 2/2/04
```

```
'and save it.
    Set myRecurrPatt = myApptItem.GetRecurrencePattern
   myRecurrPatt.RecurrenceType = olRecursDaily
   myRecurrPatt.PatternStartDate = #2/2/2003#
    myRecurrPatt.PatternEndDate = #2/2/2004#
    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 3/12/03.
    myDate = #3/12/2003 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 = #3/12/2003 3:30:00 PM#
    mvOddApptItem.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.
    MsqBox myException.OriginalDate & ": " & saveSubject
    'Display the current date, time, and subject
    'for this exception.
    MsqBox myException.AppointmentItem.Start & ": " &
   myException.AppointmentItem.Subject
End Sub
```

**Note** For this example to work properly in VBScript, a few changes need to be made in the code.

You do not 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)
```



# ExchangeConnectionMode Property

Returns an **<u>OlExchangeConnectionMode</u>** constant that indicates the current connection mode the user is using. Read-only.

**OlExchangeConnectionMode** can be one of the following constants:

- olOffline(100)
- olOnline (500)
- olDisconnected (200)
- olConnectedHeaders (300)
- olConnected (400)
- olNoExchange (0)

#### expression. ExchangeConnectionMode

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

### Remarks

The **Namespace.Offline** property will also return **True** if the connection mode is offline. However, if the connection mode is low bandwidth or online, the **Namespace.Offline** property will return **False**.

The following Microsoft Visual Basic for Applications (VBA) example marks the items that are sent with high importance for download if the connection mode is 'Connected Headers' and download state is 'Header Only' in the **Inbox** folder.

```
Sub MarkHighImportance()
 Dim myOlApp As New Outlook.Application
Dim myNamespace As Outlook.NameSpace
Dim mpfInbox As Outlook.MAPIFolder
Dim obj As Object
Dim ctr As Integer
Dim i As Integer
Set myNamespace = myOlApp.GetNamespace("MAPI")
Set mpfInbox = myNamespace.GetDefaultFolder(olFolderInbox)
 ctr = mpfInbox.Items.count
 If (myNamespace.ExchangeConnectionMode = olConnectedHeaders) Then
 For i = 1 To ctr
  Set obj = mpfInbox.Items.Item(i)
  If (obj.Importance <> olImportanceHigh And obj.DownloadState = ol
    obj.MarkForDownload = olMarkedForDownload
  End If
 Next
End If
End Sub
```

## **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 <u>MailItem</u>, <u>MeetingItem</u>, or <u>PostItem</u> object.

This Visual Basic for Applications (VBA) example uses the <u>Send</u> event and sends an item with an automatic expiration date.

```
Public WithEvents myItem As MailItem
Sub SendMyMail()
        Set myItem = Outlook.CreateItem(olMailItem)
        myItem.To = "Laura Jennings"
        myItem.Subject = "Data files information"
        myItem.Send
End Sub
Private Sub myItem_Send(Cancel As Boolean)
        myItem.ExpiryTime = #2/2/2003 4:00:00 PM#
End Sub
```

### **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 <u>Application</u> object.

The following Microsoft Visual Basic/Visual Basic for Applications (VBA) example displays the number of explorer windows that are open.

```
Dim myOlApp As New Outlook.Application
Private Sub CountExplorers()
    MsgBox "There are " & _
        myOlApp.Explorers.Count & " Explorers."
End Sub
```

If you use Microsoft Visual Basic Scripting Edition (VBScript) in a Microsoft Outlook form, you do not declare an <u>Application</u> object variable. This example shows how to perform the same task using VBScript.

The following Microsoft Visual Basic/Visual Basic for Applications (VBA) example uses the **Count** property and **Item** method of the **Selection** collection returned by the **Selection** property to display the senders of all mail items selected in the explorer that displays the **Inbox**. To run this example, you need to have at least one mail item selected in the explorer displaying the Inbox.

**Note** You might receive an error if you select items other than a mail item such as task request as the **SenderName** property does not exist for a **TaskRequestItem** object.

```
Sub GetSelectedItems()
   Dim myOlApp As New Outlook.Application
   Dim myOlExp As Outlook.Explorer
   Dim myOlSel As Outlook.Selection
   Dim MsgTxt As String
   Dim x As Integer
   MsgTxt = "You have selected items from: "
   Set myOlExp = myOlApp.Explorers.Item(1)
   If myOlExp = "Inbox" Then
   Set myOlSel = myOlExp.Selection
   For x = 1 To myOlSel.Count
```

```
MsgTxt = MsgTxt & myOlSel.Item(x).SenderName & ";"
Next x
MsgBox MsgTxt
End If
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

## **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

## **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

### Remarks

The **Filter** property is set by the *Filter* argument when the **Search** object is first created.

The following Microsoft Visual Basic/Visual Basic for Applications (VBA) 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 in addition to the results of the search.

```
Sub SearchInboxFolder()
'Searches the Inbox folder
Dim objSch As Outlook.Search
Const strF As String = _
"urn:schemas:mailheader:subject = 'Office Holiday 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
    Dim objRsts As Outlook.Results
    Dim Item as Outlook.MailItem
    MsgBox "The search " & SearchObject.Tag & "has finished. The fil
        SearchObject.Filter & "."
    Set objRsts = SearchObject.Results
    'Print out number in results collection
    MsgBox objRsts.Count
    'Print out each member of results collection
    For Each Item In objRsts
        MsgBox Item
    Next
```

```
End Sub
```

**Searching String Text fields** 

When searching **Text** fields, you can use either an apostrophe (') or double

quotation marks ("") to delimit the values that are part of the filter. For example, all of the following lines function correctly when the field is of type **String**:

sFilter = "[CompanyName] = 'Microsoft'"
sFilter = "[CompanyName] = ""Microsoft"""
sFilter = "[CompanyName] = " & Chr(34) & "Microsoft" & Chr(34)

### **FirstName Property**

Returns or sets a **String** representing the first name for the contact. Read/write.

#### expression.FirstName

### 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 an e-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.



## **FlagIcon Property**

Returns or sets an **olFlagIcon** constant indicating one or none of the six flag types in Microsoft Office Outlook 2003 for e-mail messages. Read/write.

#### object.FlagIcon

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

**olFlagIcon** can be one of the following constants:

- olNoFlagIcon (0)
- olPurpleFlagIcon (1)
- olOrangeFlagIcon (2)
- olGreenFlagIcon (3)
- olYellowFlagIcon (4)
- olBlueFlagIcon (5)
- olRedFlagIcon (6)

### Remarks

Setting the **FlagIcon** property before sending an item will not set the flag on the item when the recipient receives it.

The following Microsoft Visual Basic for Applications (VBA) example loops all items in a folder named Test in the **Inbox** and sets the yellow flag on items sent by Dan Wilson. To run this example without errors, make sure the Test folder exists in the default **Inbox** folder and replace 'Dan Wilson' with a valid sender name in the Test folder.

```
Sub SetFlagIcon()
    Dim myOlApp As Outlook.Application
    Dim mpfInbox As Outlook.MAPIFolder
    Dim obj As Outlook.MailItem
    Dim i As Integer
    Set myOlApp = CreateObject("Outlook.Application")
    Set mpfInbox = myOlApp.GetNamespace("MAPI").GetDefaultFolder(olF
 ' Loop all items in the Inbox\Test Folder
    For i = 1 To mpfInbox.Items.count
        If mpfInbox.Items(i).Class = olMail Then
            Set obj = mpfInbox.Items.Item(i)
                If obj.SenderName = "Dan Wilson" Then
                'Set the yellow flag icon
                obj.FlagIcon = olYellowFlagIcon
                obj.Save
            End If
        End If
    Next
End Sub
```

## **FlagRequest Property**

Returns or sets a **String** indicating the requested action for an e-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 **OIFlagStatus** constant indicating the flag status for an e-mail message. Read/write.

OlFlagStatus can be one of these OlFlagStatus constants. olFlagComplete olFlagMarked olNoFlag

expression.FlagStatus

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

## **FolderPath Property**

Returns a **String** that indicates the path of the current folder. Read-only.

### expression.FolderPath

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

The following example displays information about the default Contacts folder. The subroutine accepts a **MAPIFolder** object and displays the folder's name, path, and address book information.

```
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(olFolderContacts)
    '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.Folder
             ". The current address book name is " & fldFolder.Addre
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.

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

```
Sub CreatePersonalContacts()
        Dim myOlApp As Outlook.Application
        Dim myNamespace As Outlook.NameSpace
        Dim myFolder As Outlook.MAPIFolder
        Dim myNewFolder As Outlook.MAPIFolder
        Set myOlApp = CreateObject("Outlook.Application")
        Set myNamespace = myOlApp.GetNamespace("MAPI")
        Set myFolder = myNamespace.GetDefaultFolder(olFolderContacts
        Set myNewFolder = myFolder.Folders.Add("My Personal Contacts
```

End Sub

If you use Microsoft Visual Basic Scripting Edition (VBScript) in a Microsoft Outlook form, you do not create the **Application** object, and you cannot use named constants. This example shows how to perform the same task using VBScript.

```
Sub CommandButton1_Click()
Set myNameSpace = Application.GetNameSpace("MAPI")
Set myFolder = myNamespace.GetDefaultFolder(10)
Set myNewFolder = myFolder.Folders.Add("My Personal Contacts")
End Sub
```

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

```
Sub CreateFolders()
        Dim myOlApp As Outlook.Application
        Dim myNamespace As Outlook.NameSpace
        Dim myFolder As Outlook.MAPIFolder
        Dim myNotesFolder As Outlook.MAPIFolder
        Dim myContactFolder As Outlook.MAPIFolder
        Set myOlApp = CreateObject("Outlook.Application")
        Set myNamespace = myOlApp.GetNamespace("MAPI")
        Set myFolder = myNamespace.GetDefaultFolder(olFolderTasks)
```

```
On Error GoTo ErrorHandler
Set myNotesFolder = myFolder.Folders.Add("My Notes Folder",
Set myContactFolder = myFolder.Folders.Add("My Contacts Fold
Exit Sub
ErrorHandler:
MsgBox "Error creating the folder. The folder may al
Resume Next
```

End Sub



## **FormDescription Property**

Returns the **FormDescription** object that represents the form description for the specified Microsoft Outlook <u>item</u>.

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.

### Remarks

If a program tries to reference any type of recipient information by using the Outlook object model, a dialog box is displayed that asks you to confirm access to this information. You can allow access to the Address Book or recipient information for up to ten minutes after you receive the dialog box. This allows features, such as mobile device synchronization, to be completed.

You receive the confirmation dialog box when a solution tries to programmatically access the **Formula** property.

The following Visual Basic for Applications (VBA) example shows how to use the **Formula** property.

```
Sub TestFormula()
    Dim outApp As New Outlook.Application
    Dim tki As Outlook.TaskItem
    Dim uprs As Outlook.UserProperties
    Dim upr As Outlook.UserProperty
    Set tki = outApp.CreateItem(olTaskItem)
    tki.Subject = "Work hours - Test Formula"
    tki.TotalWork = 4
    tki.ActualWork = 3
    Set uprs = tki.UserProperties
   Set upr = uprs.Add("Total&ActualWork", olFormula)
    upr.Formula = "[Total Work] + [Actual Work]"
    tki.Save
    tki.Display
  MsgBox "The Work Hours are: " & upr.Value
End Sub
```

## **FTPSite Property**

Returns or sets a **String** representing the FTP site entry for the contact. Read/write.

expression.FTPSite

## **FullName Property**

Returns or sets a **String** specifying the whole, unparsed full name for the contact. Read/write.

expression.FullName

### Remarks

This property is parsed into the **FirstName**, **MiddleName**, **LastName**, and **Suffix** properties, which may be changed or typed independently if they are parsed incorrectly. Any changes or entries to the **FirstName**, **LastName**, **MiddleName**, or **Suffix** properties will be overwritten by any subsequent changes or entries to **FullName**.

This Visual Basic for Applications (VBA) example uses the **Restrict** method to apply a filter to the contact items based on the item's **LastModificationTime** property, and then it displays the full name of the contacts returned by the filter.

```
Public Sub ContactDateCheck()
    Dim myOlApp As Outlook.Application
    Dim myNamespace As Outlook.NameSpace
    Dim myContacts As Outlook. Items
    Dim myItems As Outlook.Items
    Dim myItem As Object
    Set myOlApp = CreateObject("Outlook.Application")
    Set myNamespace = myOlApp.GetNamespace("MAPI")
    Set myContacts = myNamespace.GetDefaultFolder(olFolderContacts).
    Set myItems = myContacts.Restrict("[LastModificationTime] > '01/
    For Each myItem In myItems
        If (myItem.Class = olContact) Then
            MsgBox myItem.FullName & ": " & myItem.LastModificationT
        End If
    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



## **Gender Property**

Returns or sets an <u>OlGender</u> constant indicating the gender of the contact. Read/write.

OlGender can be one of these OlGender constants. olFemale olMale olUnspecified

expression.Gender



### **GetInspector Property**

Returns an **Inspector** object that represents an inspector initialized to contain the specified <u>item</u>. 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

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

This Visual Basic for Applications (VBA) example uses the <u>GetInspector</u> property to return a new, inactive inspector for **myItem**, and then toggles the **AdaptiveMenus** property.

```
Sub DisplayAdaptiveMenus()
Dim myOlApp As Outlook.Application
Dim myItem As Outlook.MailItem
Dim myInspector As Outlook.Inspector
Set myOlApp = CreateObject("Outlook.Application")
Set myItem = myOlApp.CreateItem(olMailItem)
Set myInspector = myItem.GetInspector
myInspector.CommandBars.AdaptiveMenus = Not myInspector.CommandBars.
myInspector.Display
End Sub
```

If you use Microsoft Visual Basic Scripting Edition (VBScript) in a Microsoft Outlook form, you do not create the <u>Application</u> object, and you cannot use named constants. This example shows how to perform the same task using VBScript code.

```
Set myItem = Application.CreateItem(0)
Set myInspector = myItem.GetInspector
myInspector.CommandBars.AdaptiveMenus = Not myInspector.CommandBars.
myInspector.Display
```

## **GovernmentIDNumber Property**

Returns or sets a **String** representing the government ID number for the contact. Read/write.

### expression.GovernmentIDNumber



## **Groups Property**

Returns an **OutlookBarGroups** object representing the set of groups in the <u>Shortcuts pane</u>.

expression.Groups

*expression* Required. An expression that returns an <u>OutlookBarStorage</u> object.

The following Microsoft Visual Basic/Visual Basic for Applications (VBA) example displays the number of groups in the Shortcuts pane.

```
Sub CountOlBarGroups()
    Dim myOlApp As New Outlook.Application
    Dim myOlBar As Outlook.OutlookBarPane
    Dim myCount As Integer
    Set myOlBar = myOlApp.ActiveExplorer.Panes.Item("OutlookBar"
    myCount = myOlBar.Contents.Groups.Count
    MsgBox "There are " & myCount & " groups in the Shortcuts pa
End Sub
```

If you use Microsoft Visual Basic Scripting Edition (VBScript) in a Microsoft Outlook form, you do not create the <u>Application</u> object. This example shows how to perform the same task using VBScript code.

```
Set myOlBar = Application.ActiveExplorer.Panes.Item("OutlookBar")
myCount = myOlBar.Contents.Groups.Count
MsgBox "There are " & myCount & " groups in the Shortcuts pane"
```

## **HasAttachment Property**

**True** (default) if the remote item has an attachment associated with it. Read-only **Boolean**.

expression.HasAttachment

## **HasPicture Property**

True if a **Contacts** item has a picture associated with it. Read-only **Boolean**.

#### expression.HasPicture

### Example

The following Microsoft Visual Basic for Applications (VBA) example prompts the user to specify the name of a contact and the file name containing a picture of the contact, and then adds the picture to the contact item. If a picture already exists for the contact item, the example prompts the user to specify if the existing picture should be overwritten by the new file.

```
Sub AddPictureToAContact()
    Dim myOlApp As Outlook.Application
    Dim myNms As Outlook.NameSpace
    Dim myFolder As Outlook.MAPIFolder
    Dim myContactItem As Outlook.ContactItem
    Dim strName As String
    Dim strPath As String
    Dim strPrompt As String
    Set myOlApp = CreateObject("Outlook.Application")
    Set myNms = myOlApp.GetNamespace("MAPI")
    Set myFolder = myNms.GetDefaultFolder(olFolderContacts)
    strName = InputBox("Type the name of the contact: ")
    Set myContactItem = myFolder.Items(strName)
    If myContactItem.HasPicture = True Then
     strPrompt = MsgBox("The contact already has a picture associate
     If strPrompt = vbNo Then
     Exit Sub
     End If
    End If
    strPath = InputBox("Type the file name for the contact: ")
    myContactItem.AddPicture (strPath)
   myContactItem.Save
   myContactItem.Display
End Sub
```

# **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

# Home2TelephoneNumber Property

Returns or sets a **String** representing the second home telephone number for the contact. Read/write.

#### expression.Home2TelephoneNumber

## **HomeAddress Property**

Returns or sets a **String** representing the full, unparsed text of the home address for the contact. Read/write.

expression.HomeAddress

# **HomeAddressCity Property**

Returns or sets a **String** representing the city portion of the home address for the contact. Read/write.

#### expression.HomeAddressCity

This property is parsed from the <u>HomeAddress</u> 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

This property is parsed from the <u>HomeAddress</u> 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

This property is parsed from the <u>HomeAddress</u> 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

This property is parsed from the <u>HomeAddress</u> 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

This property is parsed from the <u>HomeAddress</u> 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

This property is parsed from the <u>HomeAddress</u> 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

# **HomeTelephoneNumber Property**

Returns or sets a **String** representing the first home telephone number for the contact. Read/write.

#### expression.HomeTelephoneNumber



# **HTMLBody Property**

Returns or sets a **String** representing the HTML body of the specified <u>item</u>. The **HTMLBody** property should be an HTML syntax string. Read/write.

Setting the **HTMLBody** property sets the **<u>EditorType</u>** property of the item's **<u>Inspector</u>** to **olEditorHTML**.

Setting the **HTMLBody** property will always update the **<u>Body</u>** property immediately.

Setting the **Body** property will clear the contents of the **HTMLBody** property on HTML-aware stores.

#### expression.HTMLBody

*expression* Required. An expression that returns a **<u>PostItem</u>** or <u>MailItem</u> object.

Outlook blocks code that attempts to access the **HTMLBody** property for security reasons. If you run a third-party add-in, custom solution, or other program that uses the **HTMLBody** property in Office Outlook 2003, you may receive the following warning:

A program is trying to access e-mail addresses you have stored in Outlook. Do you want to allow this? If this is unexpected, it may be a virus and you should choose "No".

### Example

The following Visual Basic for Applications (VBA) example creates a new **MailItem** object and sets the **BodyFormat** property to **olFormatHTML**. The body text of the e-mail item will now appear in HTML format.

```
Sub CreateHTMLMail()
'Creates a new e-mail item and modifies its properties.
Dim olApp As Outlook.Application
Dim objMail As Outlook.MailItem
Set olApp = Outlook.Application
'Create e-mail item
Set objMail = olApp.CreateItem(olMailItem)
With objMail
    'Set body format to HTML
    .BodyFormat = olFormatHTML
    .HTMLBody = "<HTML><H2>The body of this message will appear i
    .Display
End With
End Sub
```

This Microsoft Visual Basic Scripting Edition (VBScript) example uses the **Open** event to access the **HTMLBody** property of an <u>item</u>. 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 e-mail 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 uses 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.</BOD
   'Item displays HTML message.
   Item.Display
   'MsgBox shows EditorType is 2.</pre>
```

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 '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.E 'Access the item's 'FormDescription object. Set myForm = Item.FormDescription 'Display all the code 'in the Script Editor. MsqBox 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.

In order to use this property, a folder must be using a folder home page, or you can set the **WebViewURL** property of the **MAPIFolder** object to a Web page.

### Example

The following Microsoft Visual Basic/Visual Basic for Applications (VBA) example accesses the Microsoft Outlook View Control.

```
Sub GetHTML()
'Returns the Outlook View Control
Dim objVC As OLXLib.ViewCtl
Dim objExp As Outlook.Explorer
Dim HTMLDoc As MSHTML.HTMLDocument
'Reference the current folder
Set objExp = Application.ActiveExplorer
'Reference the HTML file that is the home page
Set HTMLDoc = objExp.HTMLDocument
'Reference an Outlook View Control that is on the HTML page
Set objVC = HTMLDoc.all.tags("object").Item(0).Object
'Have the control display an address book window
objVC.AddressBook
```

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 **<u>EditorType</u>** property of the item's associated **<u>Inspector</u>** is set to **olEditorHTML**.

#### expression.HTMLEditor

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

Outlook blocks code that attempts to access the **HTMLEditor** property for security reasons. If you run a third-party add-in, custom solution, or other program that uses the **HTMLEditor** property in Office Outlook 2003, you may receive the following warning:

A program is trying to access e-mail addresses you have stored in Outlook. Do you want to allow this? If this is unexpected, it may be a virus and you should choose "No".

### Example

The following Microsoft Visual Basic Scripting Edition (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
                   'As String
    Dim strHTMLText
    Dim NL 'As String
    NL = chr(10) \& chr(13)
    Set myInspector = Item.GetInspector
    Set myIExplorer = myInspector.HTMLEditor
    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
'because not all elements support OuterHTML
            strHTMLText = ": " & NL & myIExplorer.All.Item(i).ou
            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 <u>session</u> to another. Read-only.

expression.ID

*expression* Required. An expression that returns an <u>AddressEntry</u> or <u>AddressList</u> object.

## **IMAddress Property**

Returns or sets a **String** that represents a contact's Microsoft Instant Messenger address. Read/write.

#### expression.IMAddress

*expression* Required. An expression that returns a <u>ContactItem</u> object.

### Remarks

Unlike the **<u>Recipients</u>** or **<u>To</u>** properties, there is no way to verify that the **IMAddress** property contains a valid address.

Outlook blocks code that attempts to access the **Recipients** object for security reasons. If you run a third-party add-in, custom solution, or other program that uses the **Recipients** object in Office Outlook 2003, you may receive the following warning:

A program is trying to access e-mail addresses you have stored in Outlook. Do you want to allow this? If this is unexpected, it may be a virus and you should choose "No".

The following example creates a new contact and prompts the user to enter an Instant Messenger address for the contact.

```
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 Messenge
objNewContact.Save
End Sub
```



### **Importance Property**

Returns or sets an **OlImportance** constant indicating the relative importance level for the Outlook <u>item</u>. 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.

This Visual Basic for Applications (VBA) example checks if the item displayed in the topmost inspector is sent by 'Dan Wilson' with 'High' importance. If it is, then it displays a message box to the user. Before running this example, replace 'Dan Wilson' with a valid name in your address book.

```
Sub CheckSenderName
Dim myOlApp As Outlook.Application
Dim myItem As Outlook.MailItem
Set myOlApp = CreateObject("Outlook.Application")
Set myItem = myOlApp.ActiveInspector.CurrentItem
If myItem.Importance = 2 And myItem.SenderName = "Dan Wilson
MsgBox "This message is sent by your manager with Hi
End If
```

# InAppFolderSyncObject Property

Returns or sets a **Boolean** that determines if the specified folder will be synchronized with the e-mail server. If **True**, this folder will be synchronized when the "Application Folders" <u>SyncObject</u> 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 selecting the check box for this folder in the Application Folders group on 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. The "Application Folders" **SyncObject** is the only Send/Receive group that can be programmatically modified.

The following Microsoft Visual Basic/Visual Basic for Applications (VBA) example sets the **Inbox** folder to be synchronized when the "Application Folders" **SyncObject** object is synchronized. The **InAppFolderSyncObject** property is used in conjunction with the **AppFolders** property of the **SyncObjects** collection.

```
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
    'Start 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.

You should not use **Count** property of **Items** collection when iterating Items collection with **IncludeRecurrence** property set to **True**. The value of **Count** will be an undefined value.

The following Visual Basic for Applications (VBA) example displays the subject of the appointments that occur between today and tomorrow including recurring appointments.

```
Sub DemoFindNext()
    Dim myOlApp As Outlook.Application
    Dim myNameSpace As Outlook.NameSpace
    Dim tdystart As Date
    Dim tdyend As Date
    Dim myAppointments As Outlook.Items
    Dim currentAppointment As Outlook.AppointmentItem
    Set myOlApp = CreateObject("Outlook.Application")
    Set myNameSpace = myOlApp.GetNamespace("MAPI")
    tdystart = VBA.Format(Now, "Short Date")
tdyend = VBA.Format(Now + 1, "Short Date")
    Set myAppointments = myNameSpace.GetDefaultFolder(olFolderCalend
    myAppointments.Sort "[Start]"
    myAppointments.IncludeRecurrences = True
    Set currentAppointment = myAppointments.Find("[Start] >= """ & t
    While TypeName(currentAppointment) <> "Nothing"
       MsgBox currentAppointment.Subject
       Set currentAppointment = myAppointments.FindNext
Wend
End Sub
```



## **Index Property**

Returns a **Long** indicating the position of the object within the collection. The **Index** property is only valid during the current <u>session</u> 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 <u>AddressList</u>, <u>Attachment</u>, or <u>Recipient</u> 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 <u>ContactItem</u> 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 <u>Application</u> object.

This Microsoft Visual Basic example uses the <u>Inspectors</u> property and the <u>Count</u> property and <u>Item</u> method of the <u>Inspectors</u> object to display the captions of all inspector windows.

```
Private Sub CommandButton1_Click()
    Dim myOlApp As New Outlook.Application
    Dim myInspectors As Outlook.Inspectors
    Dim iCount As Integer
    Dim iCount As Integer
    Set myInspectors = myOlApp.Inspectors
    iCount = myOlApp.Inspectors.Count
    If iCount > 0 Then
        For x = 1 To iCount
            MsgBox myInspectors.Item(x).Caption
        Next x
    Else
            MsgBox "No inspector windows are open."
    End If
End Sub
```

If you use Microsoft Visual Basic Scripting Edition (VBScript) in a Microsoft Outlook form, you do not declare an <u>Application</u> object variable. This example shows how to perform the same task using VBScript code.

```
Sub CommandButton1_Click()
   Dim x
   iCount = Application.Inspectors.Count
   Set myInspectors = Application.Inspectors
   If iCount > 0 Then
      For x = 1 To iCount
        MsgBox myInspectors.Item(x).Caption
      Next
   Else
        MsgBox "No Inspector windows are 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 **<u>RecurrencePattern</u>** 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.

#### expression.InternetCodepage

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

### Remarks

The following table lists the values that are supported by the **InternetCodePage** property.

Name Character Set Code Page Arabic (ISO) iso-8859-6 28596 Arabic (Windows) 1256 windows-1256 Baltic (ISO) 28594 iso-8859-4 Baltic (Windows) windows-1257 1257 Central European (ISO) 28592 iso-8859-2 windows-1250 Central European (Windows) 1250 Chinese Simplified (GB2312) qb2312 936 Chinese Simplified (HZ) hz-qb-2312 52936 Chinese Traditional (Big5) big5 950 Cyrillic (ISO) iso-8859-5 28595 Cyrillic (KOI8-R) koi8-r 20866 Cyrillic (KOI8-U) koi8-u 21866 Cyrillic (Windows) windows-1251 1251 Greek (ISO) iso-8859-7 28597 Greek (Windows) windows-1253 1253 Hebrew (ISO-Logical) 38598 iso-8859-8-i Hebrew (Windows) windows-1255 1255 Japanese (EUC) euc-jp 51932 Japanese (JIS) iso-2022-jp 50220 Japanese (JIS-Allow 1 byte Kana) csIS02022JP 50221 Japanese (Shift-JIS) iso-2022-jp 932 Korean ks\_c\_5601-1987 949 Korean (EUC) euc-kr 51949 Latin 3 (ISO) iso-8859-3 28593 Latin 9 (ISO) iso-8859-15 28605 Thai (Windows) windows-874 874 Turkish (ISO) iso-8859-9 28599 Turkish (Windows) windows-1254 1254 Unicode (UTF-7) utf-7 65000 Unicode (UTF-8) utf-8 65001 us-ascii US-ASCII 20127 *Vietnamese (Windows)* windows-1258 1258 Western European (ISO) iso-8859-1 28591 Western European (Windows) Windows-1252 1252

The following table lists the code pages Microsoft recommends that you use for the best compatiblity with older e-mail systems.

Name

Arabic (Windows)	<pre>windows-1256</pre>	1256
Baltic (ISO)	iso-8859-4	28594
Central European (ISO)	iso-8859-2	28592
Chinese Simplified (GB2312)	gb2312	936
Chinese Traditional (Big5)	big5	950
Cyrillic (KOI8-R)	koi8-r	20866
Cyrillic (Windows)	windows-1251	1251
Greek (ISO)	iso-8859-7	28597
Hebrew (Windows)	windows-1255	1255
Japanese (JIS)	iso-2022-jp	50220
Korean	ks_c_5601-1987	949
Thai (Windows)	windows-874	874
Turkish (ISO)	iso-8859-9	28599
Unicode (UTF-8)	utf-8	65001
US-ASCII	us-ascii	20127
Vietnamese (Windows)	windows-1258	1258
Vietnamese (Windows)	windows-1258	1258
Western European (ISO)	iso-8859-1	28591

The following Microsoft Visual Basic/Visual Basic for Applications (VBA) example displays the sender name of the current e-mail item if its Internet code page value is 1256. This value corresponds to the Internet code page value for Arabic text.

## 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

*expression* Required. An expression that returns a **<u>ContactItem</u>** 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 **<u>PatternEndDate</u>**. Also, the **Interval** property is not valid for yearly recurrence patterns.

#### expression.Interval

*expression* Required. An expression that returns a **<u>RecurrencePattern</u>** object.

This Visual Basic for Applications (VBA) example uses **GetRecurrencePattern** to obtain the **RecurrencePattern** object for the newly created **AppointmentItem**. When 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 1/21/2003 until 12/21/2004 from 2:00 PM to 5:00 PM."

```
Sub CreateAppointment()
        Dim myOlApp As Outlook.Application
        Dim myApptItem As Outlook.AppointmentItem
        Dim myRecurrPatt As Outlook.RecurrencePattern
        Set myOlApp = New Outlook.Application
        Set myApptItem = myOlApp.CreateItem(olAppointmentItem)
        Set myRecurrPatt = myApptItem.GetRecurrencePattern
        myRecurrPatt.RecurrenceType = olRecursWeekly
        myRecurrPatt.DayOfWeekMask = olMonday
        myRecurrPatt.PatternStartDate = #1/21/2003 2:00:00 PM#
        myRecurrPatt.Interval = 3
        myRecurrPatt.PatternEndDate = #12/21/2004 5:00:00 PM#
        myApptItem.Subject = "Important Appointment"
        myApptItem.Save
        myApptItem.Display
        Set myOlApp = Nothing
        Set myApptItem = Nothing
        Set myRecurrPatt = Nothing
```

```
End Sub
```

If you use Microsoft Visual Basic Scripting Edition (VBScript) in a Microsoft Outlook form, you do not create the <u>Application</u> object, and you cannot use named constants. This example shows how to perform the same task using VBScript code.

```
Set myApptItem = Application.CreateItem(1)
Set myRecurrPatt = myApptItem.GetRecurrencePattern
myRecurrPatt.RecurrenceType = 1
myRecurrPatt.DayOfWeekMask = 2
myRecurrPatt.PatternStartDate = #1/21/03 2:00:00 PM#
myRecurrPatt.Interval = 3
```

```
myRecurrPatt.PatternEndDate = #12/21/04 5:00:00 PM#
myApptItem.Subject = "Important Appointment"
myApptItem.Save
myApptItem.Display
Set myApptItem = Nothing
Set myRecurrPatt = Nothing
```

# **IsConflict Property**

Returns a **Boolean** that determines if the e-mail 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.

The following Microsoft Visual Basic/Visual Basic for Applications (VBA) example creates a new mail item and attempts to send it. If the **IsConflict** property returns **True**, the item will not be sent.

```
Sub NewMail()
'Creates and tries to send a new e-mail message.
    Dim olApp As Outlook.Application
    Dim objNewMail As Outlook.MailItem
    Set olApp = New 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 mail item."
    End If
    Set olApp = Nothing
    Set objNewMail = Nothing
End Sub
```

## **ISDNNumber Property**

Returns or sets a **String** representing the ISDN number for the contact. Read/write.

expression.ISDNNumber

*expression* Required. An expression that returns a <u>ContactItem</u> object.

## **IsOnlineMeeting Property**

**True** if this is an online meeting. Read/write **Boolean**.

### expression.IsOnlineMeeting

*expression* Required. An expression that returns an <u>AppointmentItem</u> 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**.

#### expression.IsReadOnly

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

## **IsRecurring Property**

**True** if the appointment or task is a recurring appointment or task. When the **<u>GetRecurrencePattern</u>** method is used with an **<u>AppointmentItem</u>** or **<u>TaskItem</u>** object, this property is set to **True**. Read-only **Boolean**.

expression.IsRecurring

*expression* Required. An expression that returns a **TaskItem** or an **AppointmentItem** object.



## IsSharePointFolder Property

Returns a **Boolean** that determines if the folder is a Microsoft Windows SharePoint Services folder. Read-only.

#### expression. IsSharePointFolder

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

A Windows SharePoint Services folder is a custom folder in Microsoft Outlook that contains a live copy of the contact list or event list that lives on a Windows SharePoint Services Web site. The contact list maps to a **Contacts** folder in Outlook and the event list maps to a **Calendar** folder.

SharePoint folders are automatically created under the SharePoint Folders node in the <u>Navigation Pane</u> when a contact list or an event list is exported from the Windows SharePoint Services Web site.

Though Windows SharePoint Services folders work the same way as MAPIFolders, there are a few exceptions. The folders are read-only and any attempt to edit folder properties or add, edit, or remove existing items will fail. If you attempt to perform an action that tries to write data to the folder, Outlook will display the error message "SharePoint Team Services folders are read-only in Outlook."

**Note** A folder in the user's Microsoft Exchange server folder will never be a SharePoint folder, and no folder in the user's default Personal Folders file (.pst) will ever be a SharePoint folder. Typically the SharePoint folders will be under the node SharePoint Folders in the Navigation Pane.

The following Microsoft Visual Basic for Applications (VBA) example changes the **Subject** line of the appointment item displayed in the active inspector and saves the item. If the item is contained in a SharePoint folder, it displays a message to the user that the item cannot be modified. To run this example, make sure that an appointment item is displayed in the active inspector window. This example will modify the subject of the appointment item.

```
Sub ChangeItem()
'Checks if the item is contained in a SharePoint folder. If it is nc
    Dim myOlApp As New Outlook.Application
    Dim myItem As Outlook.AppointmentItem
    Dim fldFolder As Outlook.MAPIFolder
    Set myItem = myOlApp.ActiveInspector.CurrentItem
    Set fldFolder = myItem.Parent
    If fldFolder.IsSharePointFolder = True Then
        MsgBox "The item is contained in a Windows SharePoint Servic
    Else
    myItem.Subject = myItem.Subject + " Changed by VBA"
    myItem.Save
        MsgBox "The item has been changed."
    End If
End Sub
```

## **IsSynchronous Property**

Returns a **Boolean** indicating whether the search is synchronous.

#### expression.IsSynchronous

*expression* Required. An expression that returns a <u>Search</u> object.

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 <u>Search</u> object's <u>Stop</u> 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.

This collection is 0 based. In other words, the first object in the collection is accessed with an index value of (0) zero.

The following example displays the names of all properties created by the user. The subroutine DisplayUserProps accepts an <u>ItemProperties</u> collection and searches through it, displaying the names of all <u>ItemProperty</u> 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
           MsqBox "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 **<u>Reminder</u>** object.

Microsoft Outlook determines the return value of this property based on the state of the current reminder.

The following Microsoft Visual Basic/Visual Basic for Applications (VBA) 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 Outlook.Reminders
    Dim objRem As Outlook.Reminder
    Dim i As Integer
    Set olApp = New Outlook.Application
    Set objRems = olApp.Reminders
    For i = objRems.Count To 1 Step -1
        If objRems(i).IsVisible = True Then
            objRems(i).Dismiss
        End If
    Next
    Set olApp = Nothing
    Set objRems = Nothing
    Set objRem = Nothing
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 standard and userdefined properties associated with an e-mail item.

#### expression.ItemProperties

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

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.

The following Microsoft Visual Basic/Visual Basic for Applications (VBA) example returns the **ItemProperties** collection associated with a <u>MailItem</u> object.

```
Sub ItemProperty()
'Creates a new e-mail item and accesses its properties.
Dim olApp As Outlook.Application
Dim objMail As Outlook.MailItem
Dim objItems As Outlook.ItemProperties
Dim objItem As Outlook.ItemProperty
Set olApp = New Outlook.Application
'Create the e-mail item.
Set objMail = olApp.CreateItem(olMailItem)
'Create a reference to the e-mail item's properties collection.
Set objItems = objMail.ItemProperties
'Create a reference to the third e-mail item property.
Set objItem = objItems.Item(2)
MsgBox objItem.Name & " = " & objItem.Value
```

End Sub

### **Items Property**

Returns an <u>Items</u> collection as a collection of Microsoft Outlook items in the specified folder.

expression.Items

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

This Visual Basic for Applications (VBA) example uses the <u>Items</u> property to obtain the collection of <u>ContactItem</u> objects from the default **Contacts** folder.

```
Sub ContactDateCheck()
    Dim myOlApp As Outlook.Application
    Dim myNamespace As Outlook.NameSpace
    Dim myContacts As Outlook. Items
    Dim myItems As Outlook. Items
    Dim myItem As Object
    Set myOlApp = CreateObject("Outlook.Application")
    Set myNamespace = myOlApp.GetNamespace("MAPI")
    Set myContacts = myNamespace.GetDefaultFolder(olFolderContacts).
    Set myItems = myContacts.Restrict("[LastModificationTime] > '01/
    For Each myItem In myItems
        If (myItem.Class = olContact) Then
            MsgBox myItem.FullName & ": " & myItem.LastModificationT
        End If
    Next
End Sub
```

If you use Microsoft Visual Basic Scripting Edition (VBScript) in an Outlook form, you do not create the **Application** object, and you cannot use named constants. This example shows how to use the **Items** property in VBScript code.

```
Set myNameSpace = Application.GetNameSpace("MAPI")
Set myFolder = myNameSpace.GetDefaultFolder(10)
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 <u>ContactItem</u> 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 <u>ContactItem</u> 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.

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**.

The following Microsoft Visual Basic/Visual Basic for Applications (VBA) example sets the language type of all **View** objects of type **olTableView** to U.S. English.

```
Sub SetLanguage()
'Sets the language of all table views to U.S. English.
    Dim olApp As Outlook.Application
    Dim objViews As Outlook.Views
    Dim objView As Outlook.View
    Set olApp = New Outlook.Application
    Set objViews =
        olApp.GetNamespace("MAPI").GetDefaultFolder(olFolderInbox).V
    'Iterate through each view in the collection.
    For Each objView In objViews
        Debug.Print objView.Name
        'If view is of type olTableVIew then set language.
        If objView.ViewType = olTableView And objView.Standard = Fal
            objView.Language = "EN-US"
        End If
    Next objView
```

End Sub

## LanguageSettings Property

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 <u>Application</u> 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 **<u>ContactItem</u>** object.

## 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 <u>ContactItem</u> object.

# 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.

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**.

## LastFirstNoSpaceCompany 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. 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 **<u>ContactItem</u>** object.

## 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 <u>ContactItem</u> object.

# 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 **<u>ContactItem</u>** object.



# LastModificationTime Property

Returns a **Date** specifying the date and time that the Microsoft Outlook <u>item</u> 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 objects in the Applies To list.

This Visual Basic for Applications example uses the **Restrict** method to apply a filter to contact items based on the item's **LastModificationTime** property.

```
Public Sub ContactDateCheck()
    Dim myOlApp As Outlook.Application
    Dim myNamespace As Outlook.NameSpace
    Dim myContacts As Outlook. Items
    Dim myItems As Outlook. Items
    Dim myItem As Object
    Set myOlApp = CreateObject("Outlook.Application")
    Set myNamespace = myOlApp.GetNamespace("MAPI")
    Set myContacts = myNamespace.GetDefaultFolder(olFolderContacts).
    Set myItems = myContacts.Restrict("[LastModificationTime] > '01/
    For Each myItem In myItems
        If (myItem.Class = olContact) Then
            MsgBox myItem.FullName & ": " & myItem.LastModificationT
        End If
    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 ContactDateCheck2()
        Dim myOlApp As Outlook.Application
        Dim myNamespace As Outlook.NameSpace
        Dim myContacts As Outlook. Items
        Dim myItem As Outlook.Object
        Dim DateStart As Date
        Dim DateToCheck As String
        Dim myRestrictItems As Outlook.Items
        Set myOlApp = CreateObject("Outlook.Application")
        Set myNameSpace = myOlApp.GetNamespace("MAPI")
        Set myContacts = myNameSpace.GetDefaultFolder(olFolderContac
        DateStart = #01/1/2003#
        DateToCheck = "[LastModificationTime] >= """ & DateStart & "
        Set myRestrictItems = myContacts.Restrict(DateToCheck)
        For Each myItem In myRestrictItems
                 If (myItem.Class = olContact) Then
            MsgBox myItem.FullName & ": " & myItem.LastModificationT
        End If
        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 <u>ContactItem</u> object.

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

This property is parsed from the **<u>FirstName</u>** and **<u>LastName</u>** properties for the contact, which are themselves parsed from the <u>**FullName**</u> 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 <u>Explorer</u>, <u>Inspector</u>, or <u>NoteItem</u> 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 (VBA) 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.

```
Sub CountLinks()
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
Dim x As Integer
Dim msg As String
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
End Sub
```

If you use Microsoft Visual Basic Scripting Edition (VBScript) in a Microsoft Outlook form, you do not create the <u>Application</u> object, and you cannot use named constants. This example shows how to perform the same task using VBScript code.

```
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

## **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

### Example

This Visual Basic for Applications example uses **<u>CreateItem</u>** to create an appointment and uses <u>**MeetingStatus**</u> to set the meeting status to "Meeting" to turn it into a meeting request with both a required and an optional attendee.

```
Sub ScheduleMeeting()
        Dim myOlApp As Outlook.Application
        Dim myItem as AppointmentItem
        Dim myRequiredAttendee As Recipient
        Dim myOptionalAttendee As Recipient
        Dim myResourceAttendee As Recipient
        Set myOlApp = CreateObject("Outlook.Application")
        Set myItem = myOlApp.CreateItem(olAppointmentItem)
        myItem.MeetingStatus = olMeeting
        myItem.Subject = "Strategy Meeting"
        myItem.Location = "Conference Room B"
        myItem.Start = #9/24/2002 1:30:00 PM#
        myItem.Duration = 90
        Set myRequiredAttendee = myItem.Recipients.Add ("Nate Sun")
        myRequiredAttendee.Type = olRequired
        Set myOptionalAttendee = myItem.Recipients.Add ("Kevin Kenne
        myOptionalAttendee.Type = olOptional
        Set myResourceAttendee = myItem.Recipients.Add("Conference R
        myResourceAttendee.Type = olResource
        myItem.Send
```

```
End Sub
```

If you use VBScript, you do not create the **<u>Application</u>** 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

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 Outlook.NameSpace
    Dim objViews As Outlook.Views
    Dim objView As Outlook.View
    Set olApp = New Outlook.Application
    Set objName = olApp.GetNamespace("MAPI")
    Set objViews = objName.GetDefaultFolder(olFolderNotes).Views
    For Each objView In objViews
        If objView.SaveOption = olViewSaveOptionThisFolderEveryone T
           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

# MailingAddressCity Property

Returns or sets a **String** representing the city name portion of the selected mailing address of the contact. Read/write.

#### expression.MailingAddressCity

# 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

# 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

# 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

## MailingAddressState Property

Returns or sets a **String** representing the state code portion for the selected mailing address of the contact. Read/write.

#### expression.MailingAddressState

## MailingAddressStreet Property

Returns or sets a **String** representing the street address portion of the selected mailing address of the contact. Read/write.

#### expression.MailingAddressStreet

## **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



# **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 example searches through the user's **Inbox** for items that have not yet been fully downloaded. If any items are found that are not fully downloaded, a message is displayed and the item is marked for download.

```
Sub DownloadItems()
    Dim outApp As Outlook.Application
    Dim mpfInbox As Outlook.MAPIFolder
    Dim obj As Object
    Dim i As Integer
    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



## **MeetingResponseStatus Property**

Returns an **<u>OlResponseStatus</u>** 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 **<u>Recipient</u>** 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

### Example

This Visual Basic for Applications example uses **<u>CreateItem</u>** to create an appointment and uses <u>**MeetingStatus**</u> to set the meeting status to "Meeting" to turn it into a meeting request with both a required and an optional attendee.

```
Sub CreateAppt()
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
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.

```
Sub CommandButton1_Click()
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
End Sub
```

# MeetingWorkspaceURL Property

Returns the URL for the Meeting Workspace that the meeting or appointment item is linked to. Read-only. A Meeting Workspace is a shared Web site for planning the meeting and tracking the results.

#### expression.MeetingWorkspaceURL

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

## **MemberCount Property**

Returns a **Long** indicating the number of members in a distribution list. Read-only.

### expression.MemberCount

### 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.

```
Sub CheckDLs()
        Dim myOlApp As New Outlook.Application
        Dim myOlFolder As Outlook.MAPIFolder
        Dim myOlItems As Outlook. Items
        Dim myOlDistList As Outlook.DistListItem
        Dim x as Integer
        Set myOlFolder = myOlApp.GetNamespace("MAPI").GetDefaultFold
        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 m
                                myOlDistList.Display
                        End If
                End If
        Next x
End Sub
```

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

## **Members Property**

Returns an <u>AddressEntries</u> collection object representing the members of a distribution list in an address book.

A distribution list is an **<u>AddressEntry</u>** object whose **<u>DisplayType</u>** property is set to **olDistList** or **olPrivateDistList**.

The **Members** property returns **Nothing** if the **AddressEntry** is not a distribution list.

Read-only.

expression.Members

*expression* Required. An expression that returns an <u>AddressEntry</u> object.



### **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 <u>item</u> 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. Readonly 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.MiddleName

### 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 <u>item</u>. 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



## **ModifiedFormPages Property**

Returns the **Pages** collection that represents all the pages for the <u>item</u> 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 (VBA) displays the count of pages in the **ModifiedFormPages** collection. To run this example without any errors, display a contact item in the active window.

```
Sub CountModifiedFormPages()
    Dim myOlApp As New Outlook.Application
    Dim myItem As Outlook.ContactItem
    Dim myPages As Outlook.Pages
    Set myOlApp = CreateObject("Outlook.Application")
    Set myItem = myOlApp.ActiveInspector.CurrentItem
    Set myPages = myItem.GetInspector.ModifiedFormPages
    MsgBox myPages.Count
```

End Sub

## **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

*expression* Required. An expression that returns a **<u>RecurrencePattern</u>** 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 an object in the Applies To list. 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 (VBA) example uses the **Name** property to obtain the name of the folder displayed in the active explorer.

```
Sub DisplayCurrentFolderName()
   Dim myOlApp As Outlook.Application
   Dim myExplorer As Outlook.Explorer
   Dim myFolder As Outlook.MAPIFolder
   Set myOlApp = CreateObject("Outlook.Application")
   Set myExplorer = myOlApp.ActiveExplorer
   Set myFolder = myExplorer.CurrentFolder
   MsgBox myFolder.Name
End Sub
```

If you use Microsoft Visual Basic Scripting Edition (VBScript) in a Microsoft Outlook form, you do not create the <u>Application</u> object. This example shows how to perform the same task using VBScript code.

```
Sub CommandButton1_Click()
Set myExplorer = Application.ActiveExplorer
Set myFolder = myExplorer.CurrentFolder
MsgBox myFolder.Name
End Sub
```

# **NetMeetingAlias Property**

Returns or sets a **String** indicating the user's Microsoft NetMeeting ID, or alias. Read/write.

### expression.NetMeetingAlias

# **NetMeetingAutoStart Property**

**True** if this online meeting starts automatically. Read/write **Boolean**.

### expression. Net Meeting AutoStart



# NetMeetingDocPathName Property

Returns or sets a **String** representing the full path to the Microsoft Office document specified for a Microsoft <u>NetMeeting</u> online meeting. Read/write.

### expression.NetMeetingDocPathName

# **NetMeetingOrganizerAlias Property**

Returns or sets a **String** representing the alias of the meeting organizer, if this is an online meeting. Read/write.

### expression.NetMeetingOrganizerAlias

## **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

*expression* Required. An expression that returns an <u>AppointmentItem</u> or <u>ContactItem</u> 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

## **NetShowURL Property**

Returns or sets a **String** specifying the URL for a Microsoft NetShow online meeting. Read/write.

expression.NetShowURL

# 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 **<u>Reminder</u>** 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 <u>Caption</u> 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 Outlook.Reminders
    Dim objRem As Outlook.Reminder
    Dim strTitle As String
    Dim strReport As String
    Set olApp = New Outlook.Application
    Set objRems = olApp.Reminders
    strTitle = "Current Reminder Schedule:"
    strReport = ""
    'Check if any reminders exist.
    If objRems.Count = 0 Then
        MsgBox "There are no current reminders."
    Else
        For Each objRem In objRems
                'Add information to string.
                strReport = strReport & objRem.Caption & vbTab & _
                            objRem.NextReminderDate & vbCr
         Next objRem
        'Display report in dialog box
        MsgBox strTitle & vbCr & vbCr & strReport
    End If
```

End Sub

## NickName Property

Returns or sets a **String** representing the nickname for the contact. Read/write.

### expression.Nickname



# **NoAging Property**

**True** to not age the Outlook <u>item</u>. 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 **<u>RecurrencePattern</u>** object.

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**.

## **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 **<u>RecurrencePattern</u>** object.

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

# **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

### Example

The following Microsoft Visual Basic/Visual Basic for Applications (VBA) example returns **True** or **False** depending on whether the **Namespace** object is currently online.

```
Sub Off()
'Determines whether Outlook is currently offline.
   Dim olapp As Outlook.Application
   Dim nmsName As Outlook.NameSpace
   Set olapp = New Outlook.Application
   Set nmsName = olapp.GetNamespace("MAPI")
   MsgBox nmsName.Offline
End Sub
```

## **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

# **Ordinal Property**

Returns or sets a **Long** specifying the position in the view (ordinal) for the task. Read/write.

expression.Ordinal

# **OrganizationalIDNumber Property**

Returns or sets a **String** representing the organizational ID number for the contact. Read/write.

#### expression.OrganizationalIDNumber

# **Organizer Property**

Returns a **String** representing the name of the organizer of the appointment. Read-only.

expression.Organizer

# **OriginalDate Property**

Returns a **Date** indicating the original date and time of an **AppointmentItem** before it was altered. This property will return the original date even if the **AppointmentItem** has been deleted. However, 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 (VBA) 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 Microsoft Visual Basic Scripting Edition (VBScript), see the Note at the end of the example.

```
Public Sub cmdExample()
    Dim myOlApp As Outlook.Application
    Dim myApptItem As Outlook.AppointmentItem
    Dim myRecurrPatt As Outlook.RecurrencePattern
    Dim myNamespace As Outlook.NameSpace
    Dim myFolder As Outlook.MAPIFolder
    Dim myItems As Outlook. Items
    Dim myDate As Date
    Dim myOddApptItem As Outlook.AppointmentItem
    Dim saveSubject As String
    Dim newDate As Date
    Dim myException As Outlook.Exception
    Set myOlApp = New Outlook.Application
    Set myApptItem = myOlApp.CreateItem(olAppointmentItem)
   myApptItem.Start = #2/2/2003 3:00:00 PM#
   myApptItem.End = #2/2/2003 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 2/2/03 and ends on 2/2/04
```

```
'and save it.
    Set myRecurrPatt = myApptItem.GetRecurrencePattern
   myRecurrPatt.RecurrenceType = olRecursDaily
   myRecurrPatt.PatternStartDate = #2/2/2003#
    myRecurrPatt.PatternEndDate = #2/2/2004#
    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 3/12/03.
    myDate = #3/12/2003 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 = #3/12/2003 3:30:00 PM#
    mvOddApptItem.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.
    MsqBox myException.OriginalDate & ": " & saveSubject
    'Display the current date, time, and subject
    'for this exception.
    MsqBox myException.AppointmentItem.Start & ": " &
   myException.AppointmentItem.Subject
End Sub
```

**Note** For this example to work properly in VBScript, a few changes need to be made in the code.

You do not 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)
```

# **OriginalReminderDate Property**

Returns a **Date** that specifies the original date and time that the specified reminder is set to occur. Read-only.

#### expression. Original Reminder Date

*expression* Required. An expression that returns a **<u>Reminder</u>** object.

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 Microsoft Visual Basic/Visual Basic for Applications (VBA) example creates a report of all reminders in the <u>Reminders</u> collection and the dates at which they are scheduled to occur. The subroutine concatenates the <u>Caption</u> and **OriginalReminderDate** properties of all <u>Reminder</u> 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 Outlook.Reminders
    Dim objRem As Outlook.Reminder
    Dim strTitle As String
    Dim strReport As String
    Set olApp = New Outlook.Application
    Set objRems = olApp.Reminders
    strTitle = "Original Reminder Schedule:"
    strReport = ""
    'Check if any reminders exist.
    If objRems.Count = 0 Then
        MsgBox "There are no current reminders."
    Else
        For Each objRem In objRems
                'Add info to string
                strReport = strReport & objRem.Caption & vbTab & vbT
                            objRem.OriginalReminderDate & vbCr
        Next objRem
        'Display report in dialog
        MsgBox strTitle & vbCr & vbCr & 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 <u>MailItem</u> or a <u>MeetingItem</u> object.

## **OtherAddress Property**

Returns or sets a **String** representing the other address for the contact. Read/write.

expression.OtherAddress

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

This property is parsed from the <u>OtherAddress</u> 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

This property is parsed from the <u>OtherAddress</u> 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

This property is parsed from the <u>OtherAddress</u> 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

This property is parsed from the <u>OtherAddress</u> 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.OtherAddressState

This property is parsed from the <u>OtherAddress</u> 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

### Remarks

This property is parsed from the <u>OtherAddress</u> 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

# **OtherTelephoneNumber Property**

Returns or sets a **String** representing the other telephone number for the contact. Read/write.

### expression.OtherTelephoneNumber



# **OutlookInternalVersion Property**

Returns a **Long** representing the build number of the Microsoft Outlook application for an Outlook <u>item</u>. 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 <u>item</u>. Read-only.

### expression.OutlookVersion

*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 if the task is stored on the Exchange Server public folder. Read-only if it's stored in a user's mailbox or personal folders file.

#### expression.Owner



# **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

# **PagerNumber Property**

Returns or sets a **String** representing the pager number for the contact. Read/write.

expression.PagerNumber



### **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 (VBA) example adds a group named "Marketing" as the second group in the <u>Shortcuts pane</u>.

```
Sub AddGroup()
    Dim myOlApp As New Outlook.Application
    Dim myolBar As Outlook.OutlookBarPane
    Set myolBar = myOlApp.ActiveExplorer.Panes.Item("OutlookBar"
    myolBar.Contents.Groups.Add "Sales", myolBar.Contents.Groups
End Sub
```

If you use Microsoft Visual Basic Scripting Edition (VBScript) in a Microsoft Outlook form, you do not create the <u>Application</u> object. This example shows how to perform the same task using VBScript code.

Set myolBar = Application.ActiveExplorer.Panes.Item("OutlookBar")
myolBar.Contents.Groups.Add "Marketing", myolBar.Contents.Groups.Cou

### **Parent Property**

Returns the parent **Object** of the specified object. Read-only.

### expression.Parent

*expression* Required. An expression that returns a Microsoft Outlook object.

### Example

This Visual Basic for Applications (VBA) example displays the folders in the parent folder of **Inbox**.

```
Sub GetRootFolder()
   Dim mpfRoot As Outlook.MAPIFolder
   Dim mpf As Outlook.MAPIFolder
   Dim idx As Integer
   Set mpf = Application.Session.GetDefaultFolder(olFolderInbox)
   Set mpfRoot = mpf.Parent
   For idx = 1 To mpfRoot.Folders.count
        MsgBox mpfRoot.Folders.Item(idx).Name
   Next
```

End Sub

## **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

## **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 **<u>RecurrencePattern</u>** object.

### Remarks

This property is optional but must be coordinated with other properties when setting up a recurrence pattern. If this property or the <u>Occurrences</u> property is set, the pattern is considered to be finite, and the <u>NoEndDate</u> property is **False**. If neither **PatternEndDate** nor **Occurrences** is set, the pattern is considered infinite and **NoEndDate** is **True**. The <u>Interval</u> 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 **<u>RecurrencePattern</u>** 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



## **Permission Property**

Sets or returns an **<u>OlPermission</u>** constant that determines the permissions the recipients will have on the e-mail item. Read/write.

OlPermission can be one of the following:

- olUnrestricted (0)
- olDoNotForward (1)
- olPermissionTemplate (2)

#### expression.Permission

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

### Example

This Microsoft Visual Basic for Applications (VBA) example uses the <u>Send</u> event and sends an <u>item</u> with a 'Do not forward' restriction. The sample code must be placed in a class module such as **ThisOutlookSession**, and the SendMyMail procedure must be called before the event procedure can be called by Microsoft Outlook. Replace 'Dan Wilson' with a valid recipient name before running this example.

```
Public WithEvents myItem As Outlook.MailItem
Sub SendMyMail()
   Set myItem = Outlook.CreateItem(olMailItem)
   myItem.To = "Dan Wilson"
   myItem.Subject = "Data files information"
   myItem.Send
End Sub
Private Sub myItem_Send(Cancel As Boolean)
   myItem.Permission = olDoNotForward
End Sub
```



# **PermissionService Property**

Sets or returns an **OlPermissionService** constant that determines the permission service that will be used when sending an IRM protected message. Note that this property is useful only if you have more than one permission identity for a particular SMTP address. Read/write.

OlPermissionService can be one of the following:

- olUnknown (0)
- olWindows (1)
- olPassport (2)

### expression.PermissionService

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

### Example

This Microsoft Visual Basic for Applications (VBA) example demonstrates how to specify the permission service before sending an <u>item</u>. Replace 'Dan Wilson' with a valid recipient name before running this example.

```
Sub SendMyMail()
   Set myItem = Outlook.CreateItem(olMailItem)
   myItem.To = "Dan Wilson"
   myItem.Subject = "Data files information"
   myItem.PermissionService = olWindows
   myItem.Send
End Sub
```

## **PersonalHomePage Property**

Returns or sets a **String** representing the URL of the personal Web page for the contact. Read/write.

### expression.PersonalHomePage



# **Position Property**

Returns or sets a **Long** indicating the position of the attachment within the body of the <u>item</u>. Read/write.

### expression.Position

## **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

# **PrimaryTelephoneNumber Property**

Returns or sets a **String** specifying the primary telephone number for the contact. Read/write.

### expression.PrimaryTelephoneNumber

# **ProductCode Property**

Returns a **String** specifying the Microsoft Outlook <u>globally unique identifier</u> <u>(GUID</u>). Read-only.

expression.ProductCode()

*expression* Required. An expression that returns an <u>Application</u> object.

## **Profession Property**

Returns or sets a **String** indicating the profession for the contact. Read/write.

### expression.Profession

# **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 <u>ContactItem</u> 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

# **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

## **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

# 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

# **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

## **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 <u>MailItem</u>, <u>MeetingItem</u>, or <u>PostItem</u> object.

# RecipientReassignmentProhibited Property

**True** if the recipient cannot forward the mail message. Read/write **Boolean**.

### expression.RecipientReassignmentProhibited



# **Recipients Property**

Returns a **<u>Recipients</u>** collection that represents all the recipients for the Microsoft Outlook <u>item</u>. Read-only.

expression.Recipients

*expression* Required. An expression that returns an <u>AppointmentItem</u>, <u>JournalItem</u>, <u>MailItem</u>, <u>MeetingItem</u>, or <u>TaskItem</u> object.

## Remarks

Outlook blocks code that attempts to access the **Recipients** property for security reasons. If you run a third-party add-in, custom solution, or other program that uses the **Recipients** property in Office Outlook 2003, you may receive the following warning:

A program is trying to access e-mail addresses you have stored in Outlook. Do you want to allow this? If this is unexpected, it may be a virus and you should choose "No".

## Example

This Visual Basic for Applications (VBA) example creates a new e-mail message, uses the <u>Add</u> method to add "Dan Wilson" as a **To** recipient, and displays the message.

```
Sub CreateStatusReportToBoss()
    Dim myOlApp As Outlook.Application
    Dim myItem As Outlook.MailItem
    Dim myRecipient As Outlook.Recipient
    Set myOlApp = CreateObject("Outlook.Application")
    Set myItem = myOlApp.CreateItem(olMailItem)
    Set myRecipient = myItem.Recipients.Add("Dan Wilson")
    myItem.Subject = "Status Report"
    myItem.Display
End Sub
```

End Sub

If you use Microsoft Visual Basic Scripting Edition (VBScript) in an Outlook form, you do not create the <u>Application</u> object, and you cannot use named constants. This example shows how to perform the same task using VBScript code.

```
Set myItem = Application.CreateItem(0)
Set myRecipient = myItem.Recipients.Add("Dan Wilson")
myItem.Subject = "Status Report"
myItem.Display
```



## **RecurrenceState Property**

Returns an **<u>OlRecurrenceState</u>** 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

*expression* Required. An expression that returns an <u>AppointmentItem</u> object.



## **RecurrenceType Property**

Returns or sets an **OlRecurrenceType** constant specifying the frequency of occurrences for the recurrence pattern. Read/write.

OlRecursDaily olRecursMonthly olRecursMonthNth olRecursWeekly olRecursYearly olRecursYearNth

expression.RecurrenceType

*expression* Required. An expression that returns a **<u>RecurrencePattern</u>** object.

## Example

This Visual Basic for Applications (VBA) example creates a task called "Oil Change" that recurs every three months and uses the <u>Regenerate</u> property to set it to regenerate after each recurrence.

```
Sub CreateTaskOilChange()
    Dim myOlApp As Outlook.Application
    Dim myItem As Outlook.TaskItem
    Dim myPattern As Outlook.RecurrencePattern
    Set myOlApp = CreateObject("Outlook.Application")
    Set myItem = myOlApp.CreateItem(olTaskItem)
    Set myPattern = myItem.GetRecurrencePattern
    myPattern.Regenerate = True
    myPattern.Interval = 3
    myItem.Subject = "Oil Change"
    myItem.Save
    myItem.Display
Fad Cub
```

```
End Sub
```

If you use Microsoft Visual Basic Scripting Edition (VBScript) in a Microsoft Outlook form, you do not create the <u>Application</u> object, and you cannot use named constants. This example shows how to perform the same task using VBScript code.

```
Set myItem = Application.CreateItem(3)
Set myPattern = myItem.GetRecurrencePattern
myPattern.Regenerate = 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 <u>ContactItem</u> 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 **<u>RecurrencePattern</u>** object.

## Example

This Visual Basic for Applications (VBA) 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.

```
Sub CreateTaskOilChange()
    Dim myOlApp As Outlook.Application
    Dim myItem As Outlook.TaskItem
    Dim myPattern As Outlook.RecurrencePattern
    Set myOlApp = CreateObject("Outlook.Application")
    Set myItem = myOlApp.CreateItem(olTaskItem)
    Set myPattern = myItem.GetRecurrencePattern
    myPattern.RecurrenceType = olRecursMonthly
    myPattern.Interval = 3
    myItem.Subject = "Oil Change"
    myItem.Save
    myItem.Display
Fad Cub
```

```
End Sub
```

If you use Microsoft Visual Basic Scripting Edition (VBScript) in a Microsoft Outlook form, you do not create the <u>Application</u> object, and you cannot use named constants. This example shows how to perform the same task using VBScript code.

```
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 <u>AppointmentItem</u> 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 <u>AppointmentItem</u>, <u>MailItem</u>, or <u>TaskItem</u> 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 **<u>ReminderOverrideDefault</u>** property is set to **True**.

### expression.ReminderPlaySound

*expression* Required. An expression that returns an <u>AppointmentItem</u>, <u>MailItem</u>, or <u>TaskItem</u> object.

## **Reminders Property**

Returns a **<u>Reminders</u>** collection that represents all current reminders. Read-only.

### expression.Reminders

*expression* Required. An expression that returns an <u>Application</u> object.

## Example

The following example returns the **Reminders** collection and displays the captions of all reminders in the collection. If no current reminders are available, a message is displayed to the user.

```
Sub ViewReminderInfo()
'Lists reminder caption information
    Dim olApp As Outlook.Application
    Dim objRem As Outlook.Reminder
    Dim objRems As Outlook.Reminders
    Dim strTitle As String
    Dim strReport As String
    Set olApp = New Outlook.Application
    Set objRems = olApp.Reminders
    strTitle = "Current Reminders:"
    strReport = ""
    'If there are reminders, display message
    If olApp.Reminders.Count <> 0 Then
        For Each objRem In objRems
                'Add information to string
                strReport = strReport & objRem.Caption & vbCr
         Next objRem
        'Display report in dialog
        MsgBox strTitle & vbCr & vbCr & 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, e-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 creates an appointment item and sets the **ReminderSet** property before saving it.

```
Sub AddAppointment()
    Dim OutApp As Outlook.Application
    Dim apti As Outlook.AppointmentItem
    Set OutApp = CreateObject("Outlook.application")
    Set apti = OutApp.CreateItem(olAppointmentItem)
    apti.Subject = "Car Servicing"
    apti.Start = DateAdd("n", 16, Now)
    apti.End = DateAdd("n", 60, apti.Start)
    apti.ReminderSet = True
    apti.ReminderMinutesBeforeStart = 60
    apti.Save
End Out
```

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 <u>AppointmentItem</u>, <u>MailItem</u>, or <u>TaskItem</u> object.



## **ReminderTime Property**

Returns or sets a **Date** indicating the date and time at which the reminder should occur for the specified <u>item</u>. Read/write.

### expression.ReminderTime

*expression* Required. An expression that returns a <u>MailItem</u>, <u>MeetingItem</u> or <u>TaskItem</u> 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 **<u>RemoteItem</u>** object.



## **RemoteStatus Property**

Returns or sets an **<u>OlRemoteStatus</u>** 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

# **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

# **ReplyRecipients Property**

Returns a **<u>Recipients</u>** collection that represents all the reply recipient objects for the mail message.

### expression.ReplyRecipients

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

## Remarks

Outlook blocks code that attempts to access the **ReplyRecipients** property for security reasons. If you run a third-party add-in, custom solution, or other program that uses the **ReplyRecipients** property in Office Outlook 2003, you may receive the following warning:

A program is trying to access e-mail addresses you have stored in Outlook. Do you want to allow this? If this is unexpected, it may be a virus and you should choose "No".



# **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 <u>Action</u> 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 <u>AppointmentItem</u> 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 <u>AppointmentItem</u> 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 **<u>Recipient</u>** object.

## Remarks

When you run a program that uses the Microsoft Outlook object model to call the **Resolved** method, you receive a warning message. This warning message tells you that a program is trying to access the Address Book on your behalf and asks if you want to allow this.

This Visual Basic for Applications (VBA) example uses the **Resolve** method to resolve the **Recipient** object representing Dan Wilson, and then returns Dan's shared default **Calendar** folder.

```
Sub ResolveName()
        Dim myOlApp As Outlook.Application
        Dim myNamespace As Outlook.NameSpace
        Dim myRecipient As Outlook.Recipient
        Dim CalendarFolder As Outlook.MAPIFolder
        Set myOlApp = CreateObject("Outlook.Application")
        Set myNamespace = myOlApp.GetNamespace("MAPI")
        Set myRecipient = myNamespace.CreateRecipient("Dan Wilson")
        myRecipient.Resolve
        If myRecipient.Resolved Then
                Call ShowCalendar(myNamespace, myRecipient)
        End If
End Sub
Sub ShowCalendar(myNamespace, myRecipient)
        Dim CalendarFolder As Outlook.MAPIFolder
        Set CalendarFolder = _
        myNamespace.GetSharedDefaultFolder _
        (myRecipient, olFolderCalendar)
        CalendarFolder.Display
End Sub
```

If you use Microsoft Visual Basic Scripting Edition (VBScript) in an Outlook form, you do not create the <u>Application</u> object, and you cannot use named constants. This example shows how to perform the same task using VBScript code in a **CommandButton Click** event.

## **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 <u>AppointmentItem</u> 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 <u>AppointmentItem</u> object.



## **ResponseState Property**

Returns an **<u>OlTaskResponse</u>** 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 <u>TaskItem</u> object.



## **ResponseStatus Property**

Returns an **<u>OlResponseStatus</u>** 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 <u>AppointmentItem</u> 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 <u>Action</u> object.

## **Results Property**

Returns a **<u>Results</u>** collection that specifies the results of the search.

### expression.Results

*expression* Required. An expression that returns a <u>Search</u> object.

The following Visual Basic for Applications (VBA) example searches the **Inbox** for items with a subject that equals "Test" and displays the names of the senders of the e-mail items returned by the search. The AdvanceSearchComplete event procedure sets the boolean blnSearchComp to **True** when the search is complete. This boolean variable is used by the TestAdvancedSearchComplete() procedure to determine when the search is complete. The sample code must be placed in a class module, such as **ThisOutlookSession**, and the TestAdvancedSearchComplete() procedure must be called before the event procedure can be called by Microsoft Outlook.

```
Public blnSearchComp As Boolean
Private Sub Application_AdvancedSearchComplete(ByVal SearchObject As
        MsqBox "The AdvancedSearchComplete Event fired"
        blnSearchComp = True
End Sub
Sub TestAdvancedSearchComplete()
        Dim sch As Outlook.Search
        Dim rsts As Outlook.Results
        Dim i As Integer
        blnSearchComp = False
        Const strF As String = "urn:schemas:mailheader:subject = 'Te
        Const strS As String = "Inbox"
        Set sch = Application.AdvancedSearch(strS, strF)
        While blnSearchComp = False
                DoEvents
        Wend
        Set rsts = sch.Results
        For i = 1 To rsts.Count
                MsqBox rsts.Item(i).SenderName
        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 <u>TaskItem</u> 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.

This Microsoft Visual Basic/Visual Basic for Applications (VBA) example tests for the **Close** event and if the <u>item</u> has not been <u>Saved</u>, it uses the <u>Save</u> method to save the item without prompting the user.

```
Public WithEvents myItem As Outlook.MailItem
Public Sub Initalize_Handler()
    Set myItem = Application.ActiveInspector.CurrentItem
End Sub
Private Sub myItem_Close(Cancel As Boolean)
    If Not myItem.Saved Then
        myItem.Save
        MsgBox "Item was saved."
    End If
End Sub
```



# **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 <u>Add</u> 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 <u>View</u> object.

The following Microsoft Visual Basic/Visual Basic for Applications (VBA) example displays the names of all views that can be accessed by all users in the **Notes** folder.

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 Outlook.NameSpace
    Dim objViews As Outlook.Views
    Dim objView As Outlook.View
    Set olApp = New Outlook.Application
    Set objName = olApp.GetNamespace("MAPI")
    Set objViews = objName.GetDefaultFolder(olFolderNotes).Views
    For Each objView In objViews
        If objView.SaveOption = olViewSaveOptionThisFolderEveryone T
           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

# SaveSentMessageFolder Property

Returns a **MAPIFolder** object that represents the folder in which a copy of the e-mail message will be saved after being sent.

#### expression.SaveSentMessageFolder

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

This Visual Basic for Applications (VBA) example sends a reply to Dan Wilson and sets the **SaveMyPersonalItems** folder as the folder in which a copy of the item will be saved after being sent. To run this example without errors, make sure a mail item is open in the active inspector window and replace 'Dan Wilson' with a valid recipient name.

```
Sub SetSentFolder()
    Dim myItem As Outlook.MailITem
    Dim myResponse As Outlook.MailITem
    Dim mpfInbox As Outlook.MAPIFolder
    Dim mpf As Outlook.MAPIFolder
    Set mpfInbox = Application.Session.GetDefaultFolder(olFolderInbc
    Set mpf = mpfInbox.Folders.Add("SaveMyPersonalItems")
    Set myItem = Application.ActiveInspector.CurrentItem
    Set myResponse = myItem.Reply
    myResponse.To = "Dan Wilson"
    Set myResponse.SaveSentMessageFolder = mpf
    myResponse.Send
End Sub
```

# **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 **<u>TaskItem</u>** object.

## **Scope Property**

Returns a **String** that specifies the scope of the specified search. Read-only.

### expression.Scope

*expression* Required. An expression that returns a <u>Search</u> object.

## Remarks

The scope of the search is defined when the search is initiated. For more information, see the <u>AdvancedSearch</u> method.

The following Microsoft Visual Basic/Visual Basic for Applications (VBA) example 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 in addition to the results of the search.

```
Public blnSearchComp As Boolean
Private Sub Application_AdvancedSearchComplete(ByVal SearchObject As
        MsqBox "The AdvancedSearchComplete Event fired for " & Searc
        blnSearchComp = True
End Sub
Sub TestAdvancedSearchComplete()
        'List all items in the Inbox that do NOT have a flag.
        Dim objSch As Outlook.Search
        Const strF As String = "urn:schemas:httpmail:messageflag IS
        Const strS As String = "Inbox"
        Dim rsts As Outlook.Results
        Dim i As Integer
        blnSearchComp = False
        Const strF1 As String = "urn:schemas:mailheader:subject = 'T
        Const strS1 As String = "Inbox"
        Set objSch =
        Application.AdvancedSearch(Scope:=strS1, Filter:=strF1, Tag:
        While blnSearchComp = False
                DoEvents
        Wend
        Set rsts = objSch.Results
        For i = 1 To rsts.Count
                MsgBox rsts.Item(i).SenderName
        Next
End Sub
```

# ScriptText Property

Returns a **String** containing all the VBScript code in the form's Script Editor. Read-only.

expression.ScriptText

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

This Microsoft Visual Basic Scripting Edition (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 e-mail 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 uses 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.</BOD
    '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
    '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.E
    'Access the items's
    'FormDescription object.
    Set myForm = Item.FormDescription
    'Display all the code
    'in the Script Editor.
    MsqBox 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 <u>Search</u> object.

## Remarks

If **True**, the **Search** object searches through any subfolders in the specified filter path.

The following Microsoft Visual Basic/Visual Basic for Applications (VBA) example creates a **Search** object. The user's **Inbox** is specified as the scope of the search and the **SearchSubFolders** property is set to **True**. The event subroutine fires when the search has completed and displays the **Tag** and **Scope** properties for the new object as well as the results of the search.

```
Public blnSearchComp As Boolean
Private Sub Application_AdvancedSearchComplete(ByVal SearchObject As
        MsqBox "The AdvancedSearchComplete Event fired for " & Searc
        blnSearchComp = True
End Sub
Sub TestAdvancedSearchComplete()
        'List all items in the Inbox that do NOT have a flag:
        Dim objSch As Outlook.Search
        Const strF As String = "urn:schemas:httpmail:messageflag IS
        Const strS As String = "Inbox"
        Dim rsts As Outlook.Results
        Dim i As Integer
        blnSearchComp = False
        Const strF1 As String = "urn:schemas:mailheader:subject = 'T
        Const strS1 As String = "Inbox"
        Set objSch =
        Application.AdvancedSearch(Scope:=strS1, Filter:=strF1, Sear
        While blnSearchComp = False
                DoEvents
        Wend
        Set rsts = objSch.Results
        For i = 1 To rsts.Count
                MsgBox rsts.Item(i).SenderName
        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 <u>ContactItem</u> object.

# **Selection Property**

Returns a <u>Selection</u> 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 Microsoft Outlook Today or any folder with a Web view is currently displayed, this property returns an empty collection.

Also if a group header is selected, the **Count** property on the selection returns zero.

The following Microsoft Visual Basic/Visual Basic for Applications (VBA) 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. To run this example without any errors, select mail items only. Selecting an item that does not have the **SenderName** property will result in an error.

```
Sub GetSelectedItems()
        Dim myOlApp As New Outlook.Application
        Dim myOlExp As Outlook.Explorer
        Dim myOlSel As Outlook.Selection
        Dim MsgTxt As String
        Dim x As Integer
        MsgTxt = "You have selected items from: "
        Set myOlExp = myOlApp.ActiveExplorer
        Set myOlSel = myOlExp.Selection
        For x = 1 To myOlSel.Count
                MsgTxt = MsgTxt & myOlSel.Item(x).SenderName & ";"
        Next x
        MsgBox MsgTxt
```

```
End Sub
```

If you use Microsoft Visual Basic Scripting Edition (VBScript) in an Outlook form, you do not create the **Application** object. This example shows how to perform the same task using VBScript code.

```
Sub CommandButton1_Click()
MsqTxt = "You have selected items from: "
Set myOlSel = Application.ActiveExplorer.Selection
For x = 1 To myOlSel.Count
     MsqTxt = MsqTxt & myOlSel.Item(x).SenderName & ";"
Next x
MsgBox MsgTxt
End Sub
```

## SenderEmailAddress Property

Returns a **String** that represents the e-mail address of the sender of the e-mail message, meeting item, or post. This property corresponds to the MAPI property **PR\_SENDER\_EMAIL\_ADDRESS**. Read-only.

#### expression.SenderEmailAddress

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

### Remarks

Microsoft Outlook blocks code that attempts to access the **SenderEmailAddress** property for security reasons. If you run a third-party add-in, custom solution, or other program that uses the **SenderEmailAddress** property in Microsoft Office Outlook 2003, you may receive the following warning:

A program is trying to access e-mail addresses you have stored in Outlook. Do you want to allow this? If this is unexpected, it may be a virus and you should choose "No".

The following Microsoft Visual Basic for Applications (VBA) example loops all items in a folder named Test in the **Inbox** and sets the yellow flag on items sent by 'someone@example.com'. To run this example without errors, make sure the Test folder exists in the default **Inbox** folder and replace 'someone@example.com' with a valid sender e-mail address in the Test folder.

```
Sub SetFlagIcon()
 Dim myOlApp As Outlook.Application
 Dim mpfInbox As Outlook.MAPIFolder
 Dim obj As Outlook.MailItem
 Dim i As Integer
 Set myOlApp = CreateObject("Outlook.Application")
 Set mpfInbox = myOlApp.GetNamespace("MAPI").GetDefaultFolder(olFold
 ' Loop all items in the Inbox\Test Folder
 For i = 1 To mpfInbox.Items.Count
  If mpfInbox.Items(i).Class = olMail Then
   Set obj = mpfInbox.Items.Item(i)
    If obj.SenderEmailAddress = "someone@example.com" Then
    'Set the yellow flag icon
    obj.FlagIcon = olYellowFlagIcon
    obi.Save
   End If
  End If
Next
End Sub
```

# SenderEmailType Property

Returns a **String** that represents the type of entry for the e-mail address of the sender of the message, meeting item, or post, such as 'SMTP' for Internet address, 'EX' for a Microsoft Exchange server address, etc. Read-only.

#### expression.SenderEmailType

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

The following Microsoft Visual Basic for Applications (VBA) example demonstrates how to use the **SenderEmailType** property. To run this example without errors, an e-mail item should be open in the active inspector window.

```
Sub SenderEmailTypeExample()
   Dim myOlApp As New Outlook.Application
   Dim mail As Outlook.MailItem
   Set mail = myOlApp.ActiveInspector.CurrentItem
   MsgBox mail.SenderEmailType
   If mail.SenderEmailType = "SMTP" Then
        MsgBox "Message from Internet e-mail user."
   Else
   If mail.SenderEmailType = "EX" Then
        MsgBox "Message from internal Exchange user."
   End If
   End If
End Sub
```

## **SenderName Property**

Returns a **String** indicating the display name of the sender for the e-mail message, meeting item, or post. This property corresponds to the MAPI property PR\_SENDER\_NAME. Read-only.

**Note** If you wish to retrieve the fully qualified e-mail address of the sender, use the **SenderEmailAddress** property.

#### expression.SenderName

*expression* Required. An expression that returns a <u>MailItem</u>, <u>MeetingItem</u>, or <u>PostItem</u> object.

### Remarks

Outlook blocks code that attempts to access the **SenderName** property for security reasons. If you run a third-party add-in, custom solution, or other program that uses the **SenderName** property in Office Outlook 2003, you may receive the following warning:

A program is trying to automatically send e-mail on your behalf. Do you want to allow this? If this is unexpected, it may be a virus and you should choose "No".

This Visual Basic for Applications (VBA) example checks if the item displayed in the topmost inspector is sent by 'Dan Wilson' with 'High' importance. If it is, then it displays a message box to the user. Before running this example, replace 'Dan Wilson' with a valid name in your address book.

```
Sub CheckSenderName
Dim myOlApp As Outlook.Application
Dim myItem As Outlook.MailItem
Set myOlApp = CreateObject("Outlook.Application")
Set myItem = myOlApp.ActiveInspector.CurrentItem
If myItem.Importance = 2 And myItem.SenderName = "Dan Wilson
MsgBox "This message is sent by your manager with Hi
End If
```



# **Sensitivity Property**

Returns or sets an **OlSensitivity** constant indicating the sensitivity for the Microsoft Outlook item. This property corresponds to the MAPI property PR\_SENSITIVITY. Read/write.

OlSensitivity can be one of these OlSensitivity 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 <u>MeetingItem</u> or <u>MailItem</u> 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 <u>Send</u> method, the transport provider sets the <u>ReceivedTime</u> and **SentOn** properties for you. Read-only.

#### expression.SentOn

*expression* Required. An expression that returns a <u>MailItem</u>, <u>MeetingItem</u> or <u>PostItem</u> 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 **<u>NameSpace</u>** object for the current <u>session</u>.

Note The Session property and the GetNamespace method can be used interchangeably to obtain the Namespace object for the current session. Both members serve the same purpose. For example, the following statements do the same function: Set objNamespace = Application.GetNamespace("MAPI") Set objSession = Application.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.

The following Visual Basic for Applications (VBA) example shows how to use the **Session** property.

```
Sub GetSession()
Dim outApp As Outlook.Application
Dim nsp As Outlook.NameSpace
Set outApp = CreateObject("Outlook.Application")
Set nsp = outApp.Session
```

End Sub



# **Shortcuts Property**

Returns an **<u>OutlookBarShortcuts</u>** collection of <u>shortcuts</u> contained within the <u>Shortcuts pane</u>.

expression.Shortcuts

*expression* Required. An expression that returns an <u>OutlookBarGroup</u> object.

This Microsoft Visual Basic/Visual Basic for Applications (VBA) example deletes all empty groups in the Shortcuts pane.

```
Sub DeleteEmptyGroups()
Dim myOlApp As New Outlook.Application
Dim myOlBar As Outlook.OutlookBarPane
Dim myOlGroup As Outlook.OutlookBarGroup
Dim x As Integer
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
End Sub
```

If you use Microsoft Visual Basic Scripting Edition (VBScript) in a Microsoft Outlook form, you do not create the <u>Application</u> object. This example shows how to perform the same task using VBScript code.

```
Sub CommandButton1_Click()
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
End Sub
```

# ShowAsOutlookAB Property

Returns or sets a Boolean variable that specifies whether the contact items folder will be displayed as a Microsoft Outlook Address Book. Read-write.

#### expression.ShowAsOutlookAB

*expression* Required. An expression that returns a **MAPIFolder** object representing a contact items folder.

## Remarks

If you set the **ShowAsOutlookAB** property of a contact items folder to **False**, it will not be available in the **Show Names from the:** list in the **Select Names** dialog box.

The following Visual Basic for Applications (VBA) example creates a reference to the default **Contacts** folder and modifies its **ShowAsOutlookAB** property to be displayed as an Address Book.

```
Sub ShowAsAddressBookChange()
Dim olApp As Outlook.Application
Dim nmsName As Outlook.Namespace
Dim fldFolder As Outlook.MAPIFolder
Set olApp = Outlook.Application
'Create instance of namespace
Set nmsName = olApp.GetNamespace("Mapi")
Set fldFolder = nmsName.GetDefaultFolder(olFolderContacts)
'Display the folder as Outlook Address Book
fldFolder.ShowAsOutlookAB = True
```

End Sub



# **ShowItemCount Property**

Sets or returns an **OlShowItemCount** constant that indicates whether to display the number of unread messages in the folder or the total number of items in the folder in the Navigation Pane. Read/write.

The **OlShowItemCount** constant can be one of the following:

- olShowNoItemCount (0)
- olShowUnreadItemCount (1)
- olShowTotalItemCount (2)

### expression.ShowItemCount

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

### Remarks

The **ShowItemCount** property does not work with public folders.

This Microsoft Visual Basic for Applications (VBA) example displays the unread count for the **Inbox** in the Navigation Pane.

```
Sub ShowTotalItemCount()
Dim olApp As New Outlook.Application
Dim nmsName As Outlook.NameSpace
Dim fldFolder As Outlook.MAPIFolder
Set nmsName = olApp.GetNamespace("MAPI")
Set fldFolder = nmsName.GetDefaultFolder(olFolderInbox)
fldFolder.ShowItemCount = olShowUnreadItemCount
End Sub
```



# **ShowOn Property**

Returns or sets an **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 <u>Action</u> 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 <u>ContactItem</u> 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 <u>View</u> 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 <u>AppointmentItem</u> or <u>JournalItem</u> object.

This Visual Basic for Applications (VBA) example uses <u>CreateItem</u> to create an appointment and uses <u>MeetingStatus</u> to set the meeting status to "Meeting" and to make it a meeting request with both a required and an optional attendee.

```
Sub ScheduleMeeting()
```

```
Dim myOlApp As Outlook.Application
Dim myItem as Outlook.AppointmentItem
Dim myRequiredAttendee As Outlook.Recipient
Dim myOptionalAttendee As Outlook.Recipient
Dim myResourceAttendee As Outlook.Recipient
Set myOlApp = CreateObject("Outlook.Application")
Set myItem = myOlApp.CreateItem(olAppointmentItem)
myItem.MeetingStatus = olMeeting
myItem.Subject = "Strategy Meeting"
myItem.Location = "Conference Room B"
myItem.Start = #9/24/2003 1:30:00 PM#
myItem.Duration = 90
Set myRequiredAttendee = myItem.Recipients.Add ("Nate Sun")
myRequiredAttendee.Type = olRequired
Set myOptionalAttendee = myItem.Recipients.Add ("Kevin Kenne
myOptionalAttendee.Type = olOptional
Set myResourceAttendee = myItem.Recipients.Add("Conference R
myResourceAttendee.Type = olResource
myItem.Send
```

End Sub

If you use Microsoft Visual Basic Scripting Edition (VBScript) in a Microsoft Outlook form, you do not create the <u>Application</u> object, and you cannot use named constants. This example shows how to perform the same task using VBScript code.

```
Sub CommandButton1_Click()
Set myItem = Application.CreateItem(1)
myItem.MeetingStatus = 1
myItem.Subject = "Strategy Meeting"
myItem.Location = "Conference Room B"
myItem.Start = #9/24/03 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
End Sub
```

## **StartDate Property**

Returns or sets a **Date** specifying the starting date and time for the specified task. Read/write.

expression.StartDate

*expression* Required. An expression that returns a **<u>TaskItem</u>** 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

*expression* Required. An expression that returns a **<u>RecurrencePattern</u>** 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 **<u>TaskItem</u>** 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 **<u>TaskItem</u>** 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.

This Visual Basic for Applications (VBA) example obtains the **EntryID** and **StoreID** for the default Tasks folder and then calls the **GetFolderFromID** method using these values to obtain the same folder. The folder is then displayed.

```
Sub GetWithID()
Dim myOlApp As Outlook.Application
Dim myFolder As Outlook.MAPIFolder
Dim myEntryID As String
Dim myStoreID As String
Dim myNewFolder As Outlook.MAPIFolder
Set myOlApp = CreateObject("Outlook.Application")
Set myFolder = myOlApp.Session.GetDefaultFolder(olFolderTasks)
myEntryID = myFolder.EntryID
myStoreID = myFolder.StoreID
Set myNewFolder = myOlApp.Session.GetFolderFromID(myEntryID, myStor
myNewFolder.Display
End Sub
```

If you use Microsoft Visual Basic Scripting Edition (VBScript) in a Microsoft Outlook form, you do not create the <u>Application</u> object, and you cannot use named constants. This example shows how to use the **StoreID** property using VBScript code.

```
Sub CommandButton1_Click()
Set myNameSpace = Application.GetNameSpace("MAPI")
Set myFolder = myNameSpace.GetDefaultFolder(13)
Set myOwnFolder = myFolder.Folders("MySubFolder")
myEntryID = myOwnFolder.EntryID
myStoreID = myOwnFolder.StoreID
Set myNewFolder = myNameSpace.GetFolderFromID(myEntryID, myStoreID)
myNewFolder.Display
End Sub
```

# **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 **<u>NoteItem</u>** 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.

This Microsoft Visual Basic for Applications (VBA) example creates a new email message, uses the <u>Add</u> method to add "Dan Wilson" as a **To** recipient, sets the **Subject** property, and displays the message.

```
Sub CreateStatusReportToBoss()
    Dim myOlApp As Outlook.Application
    Dim myItem As Outlook.MailItem
    Dim myRecipient As Outlook.Recipient
    Set myOlApp = CreateObject("Outlook.Application")
    Set myItem = myOlApp.CreateItem(olMailItem)
    Set myRecipient = myItem.Recipients.Add("Dan Wilson")
    myItem.Subject = "Status Report"
    myItem.Display
End Sub
```

End Sub

If you use Microsoft Visual Basic Scripting Edition (VBScript) in a Microsoft Outlook form, you do not create the <u>Application</u> object, and you cannot use named constants. This example shows how to perform the same task using VBScript code.

```
Sub CommandButton1_Click()
Set myItem = Application.CreateItem(0)
Set myRecipient = myItem.Recipients.Add("Dan Wilson")
myItem.Subject = "Status Report"
myItem.Display
End Sub
```



# **Submitted Property**

Returns a **Boolean** value that indicates if the item has been <u>submitted</u>. **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 <u>MeetingItem</u> or <u>MailItem</u> 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 **<u>SyncObjects</u>** collection containing all Send\Receive groups. Readonly.

expression.SyncObjects

*expression* Required. An expression that returns a **<u>NameSpace</u>** object.

This Microsoft Visual Basic/Visual Basic for Applications (VBA) example displays all the Send\Receive groups set up for the user and starts the synchronization based on the user's response.

```
Public Sub Sync()
        Dim nsp As Outlook.NameSpace
        Dim sycs As Outlook.SyncObjects
        Dim syc As Outlook.SyncObject
        Dim i As Integer
        Dim strPrompt As Integer
        Set nsp = Application.GetNamespace("MAPI")
        Set sycs = nsp.SyncObjects
        For i = 1 To sycs.Count
                Set syc = sycs.Item(i)
                strPrompt = MsgBox("Do you wish to synchronize " & s
                If strPrompt = vbYes Then
                        syc.Start
                End If
        Next
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 <u>Search</u> object.

### Remarks

The **Tag** property is set by using the <u>AdvancedSearch</u> method when the <u>Search</u> object is created.

The following Visual Basic for Applications (VBA) 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. The AdvanceSearchComplete event procedure sets the boolean blnSearchComp to **True** when the search is complete. This boolean variable is used by the TestAdvancedSearchComplete() procedure to determine when the search is complete. The sample code must be placed in a class module such as **ThisOutlookSession**, and the TestAdvancedSearchComplete() sub routine must be called before the event procedure can be called by Microsoft Outlook. The AdvanceSearchComplete event procedure displays the tag to the user so the user can identify which search was completed because usually the search is asynchronous (use the **IsSynchronous** property to determine if the search will be synchronous or asynchronous), and you can execute multiple searches simultaneously.

```
Public blnSearchComp As Boolean
Private Sub Application_AdvancedSearchComplete(ByVal SearchObject As
        MsgBox "The AdvancedSearchComplete Event fired for " & Searc
        blnSearchComp = True
End Sub
Sub TestAdvancedSearch111Complete()
        'List all items in the Inbox that do NOT have a flag:
        Dim objSch As Outlook.Search
        Const strF As String = "urn:schemas:httpmail:messageflag IS
        Const strS As String = "Inbox"
        Dim rsts As Outlook.Results
        Dim i As Integer
        blnSearchComp = False
        Const strF1 As String = "urn:schemas:mailheader:subject = 'T
        Const strS1 As String = "Inbox"
        Set objSch =
        Application.AdvancedSearch(Scope:=strS1, Filter:=strF1, Tag:
        While blnSearchComp = False
                DoEvents
        Wend
        Set rsts = objSch.Results
        For i = 1 To rsts.Count
                MsgBox rsts.Item(i).SenderName
        Next
```

End Sub

## **Target Property**

Returns a **Variant** indicating the target of the specified shortcut in a Shortcuts pane group. Read-only.

expression.Target

*expression* Required. An expression that returns an <u>OutlookBarShortcut</u> 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**.

This Microsoft Visual Basic/Visual Basic for Applications (VBA) example steps through the shortcuts in the first Shortcuts pane group. If it finds a shortcut that is not an Outlook folder, it deletes it.

```
Sub DeleteShortcuts()
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
Dim myTop As Integer
Dim x As Integer
Set myOlBar = myOlApp.ActiveExplorer.Panes.Item("OutlookBar")
Set myolGroup = myOlBar.Contents.Groups.Item(1)
Set myOlShortcuts = myolGroup.Shortcuts
myTop = myOlShortcuts.Count
'Prompt the user for confirmation
Dim strPrompt As String
strPrompt = "Are you sure you want to remove all the shortcuts on th
If MsgBox(strPrompt, vbYesNo + vbQuestion) = vbYes Then
    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
End If
End Sub
```

If you use Microsoft Visual Basic Scripting Edition (VBScript) in an Outlook form, you do not create the **Application** object. This example shows how to perform the same task using VBScript code.

```
Sub CommandButton1_Click()
Set myOlBar = _
Application.ActiveExplorer.Panes.Item("OutlookBar")
Set myolGroup = myOlBar.Contents.Groups.Item(1)
Set myOlShortcuts = myolGroup.Shortcuts
myTop = myOlShortcuts.Count
'Prompt the user for confirmation
Dim strPrompt
strPrompt = "Are you sure you want to remove all the shortcuts on t
```

### **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

## **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

# **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 **<u>OlTrackingStatus</u>** constant indicating the tracking status for the recipient. Read/write.

OlTrackingStatus can be one of these OlTrackingStatus constants. olTrackingDelivered olTrackingNotDelivered olTrackingNotRead olTrackingRead olTrackingRecallFailure olTrackingRecallSuccess olTrackingReplied

expression.TrackingStatus

*expression* Required. An expression that returns a **<u>Recipient</u>** 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 **<u>Recipient</u>** object.

### **TransferSize Property**

Returns a **Long** specifying the transfer size (in bytes) for the remote item. Readonly.

expression.TransferSize

### **TransferTime Property**

Returns a **Long** indicating the transfer time (in seconds) for the remote item. Read-only.

#### expression.TransferTime

# **TTYTDDTelephoneNumber Property**

Returns or sets a **String** specifying the TTY/TDD telephone number for the contact. Read/write.

expression.TTYTDDTelephoneNumber



# **Type Property**

Type property as it applies to the **Attachment** object.

Returns an **<u>OlAttachmentType</u>** 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** or **Conflict** object.

Returns an **<u>OlObjectClass</u>** constant indicating the type of item represented by the **Link** or **Conflict** object. Read-only.

OlObjectClass can be one of these OlObjectClass constants.

olAction olActions olAddressEntries olAddressEntry olAddressList olAddressLists olApplication olAppointment olAttachments olConflict olConflicts 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

olOutlookBarShortcuts olOutlookBarStorage olPages olPanes olPost olPropertyPages olPropertyPageSite olRecipient olRecipients olRecurrencePattern olReminder olReminders olRemote olReport olResults olSearch olSelection olSyncObject olSyncObjects olTask olTaskRequest olTaskRequestAccept olTaskRequestDecline olTaskRequestUpdate olUserProperties olUserProperty olView olViews

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 olFormula olKeywords olNumber olOutlookInternal olPercent olText olYesNo

expression.**Type** 

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

Type property as it applies to the **Recipient** object.

Depending 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 (VBA) example uses <u>CreateItem</u> to create an appointment and uses <u>MeetingStatus</u> to set the meeting status to "Meeting" to turn it into a meeting request with both a required and an optional attendee. The recipient names should be replaced with valid names to avoid errors.

```
Sub ScheduleMeeting()
Dim myOlApp As Outlook.Application
Dim myItem as Outlook.AppointmentItem
Dim myRequiredAttendee As Outlook.Recipient
Dim myOptionalAttendee As Outlook.Recipient
```

```
Dim myResourceAttendee As Outlook.Recipient
Set myOlApp = CreateObject("Outlook.Application")
Set myItem = myOlApp.CreateItem(olAppointmentItem)
myItem.MeetingStatus = olMeeting
myItem.Subject = "Strategy Meeting"
myItem.Location = "Conference Room B"
myItem.Start = #9/24/2003 1:30:00 PM#
myItem.Duration = 90
Set myRequiredAttendee = myItem.Recipients.Add ("Nate Sun")
myRequiredAttendee.Type = olRequired
Set myOptionalAttendee = myItem.Recipients.Add ("Kevin Kenne
myOptionalAttendee.Type = olOptional
Set myResourceAttendee = myItem.Recipients.Add("Conference R
myResourceAttendee.Type = olResource
myItem.Send
```

If you use Microsoft Visual Basic Scripting Edition (VBScript) in a Microsoft Outlook form, you do not create the <u>Application</u> object, and you cannot use named constants. This example shows how to perform the same task using VBScript code.

End Sub

```
Sub CommandButton1_Click()
Set myItem = Application.CreateItem(1)
myItem.MeetingStatus = 1
myItem.Subject = "Strategy Meeting"
myItem.Location = "Conference Room B"
myItem.Start = #9/24/03 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
End Sub
```

# **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

### **User2 Property**

Returns or sets a **String** specifying the second Microsoft Schedule+ user for the contact. Read/write.

expression.User2

### **User3 Property**

Returns or sets a **String** specifying the third Microsoft Schedule+ user for the contact. Read/write.

expression.User3

### **User4 Property**

Returns or sets a **String** specifying the fourth Microsoft Schedule+ user for the contact. Read/write.

expression.User4

# **UserCertificate Property**

This property is not functional and is not intended for use.

### **UserProperties Property**

Returns the <u>UserProperties</u> collection that represents all the user properties for the Microsoft Outlook item.

#### expression.UserProperties

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

### Remarks

Even though **olWordDocumentItem** is a valid **OlItemType** constant, userdefined fields cannot to be added to a **DocumentItem** object and you will receive an error when you try to programmatically add a user-defined field to a **DocumentItem** object.

Note

### Example

This Visual Basic for Applications (VBA) example finds a custom property named "**LastDateContacted**" for the contact 'Jeff Smith' and displays it to the user. To run this example, you need to replace '**Jeff Smith**' with a valid contact name and create a user-defined property called **LastDateContacted** for the contact.

```
Sub FindContact()
'Finds and displays last contacted info for a contact
    Dim olApp As Outlook.Application
    Dim objContact As Outlook.ContactItem
    Dim objContacts As Outlook.MAPIFolder
    Dim objNameSpace As Outlook.NameSpace
    Dim objProperty As Outlook.UserProperty
    Set olApp = CreateObject("Outlook.Application")
    Set objNameSpace = olApp.GetNamespace("MAPI")
    Set objContacts = objNameSpace.GetDefaultFolder(olFolderContacts
    Set objContact = objContacts.Items.Find("[FileAs] = ""Smith, Jef
    If Not TypeName(objContact) = "Nothing" Then
        Set objProperty = objContact.UserProperties.Find("LastDateCo
        If TypeName(objProperty) <> "Nothing" Then
            MsgBox "Last Date Contacted: " & objProperty.Value
        End If
    Else
        MsgBox "The contact was not found."
    End If
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. Read/write.

#### expression.ValidationFormula

*expression* Required. An expression that returns a <u>UserProperty</u> object.

### Example

End Sub

The following Visual Basic for Applications (VBA) example demonstrates the use of **ValidationText** and **ValidationFormula** properties.

```
Sub TestValidation()
    Dim outApp As New Outlook.Application
    Dim tki As Outlook.TaskItem
    Dim uprs As Outlook.UserProperties
    Dim upr As Outlook.UserProperty
    Set tki = outApp.CreateItem(olTaskItem)
    tki.Subject = "Work hours"
    tki.TotalWork = 3000
    Set uprs = tki.UserProperties
    Set upr = uprs.Add("TotalWork", olFormula)
    upr.Formula = "[Total Work]"
    upr.ValidationFormula = ">= 2400"
    upr.ValidationText = """The WorkHours (Total Work) should be equ
    tki.Save
    tki.Display
  MsgBox "The Work Hours are: " & upr.Value
```

# 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 <u>UserProperty</u> object.

### Example

The following Visual Basic for Applications (VBA) example demonstrates the use of **ValidationText** and **ValidationFormula** properties.

```
Sub TestValidation()
    Dim outApp As New Outlook.Application
    Dim tki As Outlook.TaskItem
    Dim uprs As Outlook.UserProperties
    Dim upr As Outlook.UserProperty
    Set tki = outApp.CreateItem(olTaskItem)
    tki.Subject = "Work hours"
    tki.TotalWork = 3000
    Set uprs = tki.UserProperties
    Set upr = uprs.Add("TotalWork", olFormula)
    upr.Formula = "[Total Work]"
    upr.ValidationFormula = ">= 2400"
    upr.ValidationText = """The WorkHours (Total Work) should be equ
    tki.Save
    tki.Display
  MsgBox "The Work Hours are: " & upr.Value
```

End Sub

# **Value Property**

Returns or sets a **Variant** indicating the value for the specified user or item property. Read/write.

expression.Value

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

#### Example

The following Visual Basic for Applications (VBA) example creates a contact item and sets its **Body** property.

```
Sub ValueItemProperty()
   Dim outApp As New Outlook.Application
   Dim cti As Outlook.ContactItem
   Dim itms As Outlook.ItemProperties
   Dim itm As Outlook.ItemProperty
   Set cti = outApp.CreateItem(olContactItem)
   cti.FullName = "Dan Wilson"
   Set itms = cti.ItemProperties
   Set itm = itms.Item("Body")
   itm.Value = "My friend from school"
   cti.Save
   cti.Display
End Sub
```

# **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 the **Views** collection object of the **MAPIFolder** object.

#### expression.Views

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

The following Microsoft Visual Basic/Visual Basic for Applications (VBA) example creates an instance of the **Views** collection and displays the XML definition of a view called "Table View". If the view does not exist, it creates one.

```
Sub DisplayViewDef()
'Displays the XML definition of a View object
    Dim olApp As Outlook.Application
    Dim objName As Outlook.NameSpace
    Dim objViews As Outlook.Views
    Dim objView As Outlook.View
    Set olApp = New Outlook.Application
    Set objName = olApp.GetNamespace("MAPI")
    Set objViews = objName.GetDefaultFolder(olFolderInbox).Views
    'Return a view called Table View if it already exists, else crea
    Set objView = objViews.Item("Table View")
    If objView Is Nothing Then
          Set objView = objViews.Add("Table View", olTableView, olVi
    End If
    MsgBox objView.XML
End Sub
```



# ViewType Property

ViewType property as it applies to the OutlookBarGroup object.

This property does not have any effect on the icons displayed in the <u>Shortcuts</u> pane. Large icons have been removed and if this property is set to **olLargeIcon**, it will not have any effect. In previous versions of Microsoft Outlook, it returns or sets the icon view displayed by the specified Outlook Bar group. Read/write <u>OlOutlookBarViewType</u>.

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 <u>OlViewType</u> 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 <u>View</u> object.

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

This property does not have any effect on the icons displayed in the Shortcuts pane in Office Outlook 2003. Large icons have been removed in Office Outlook 2003 and if this property is set to **olLargeIcon**, it will not have any effect.

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

The following Visual Basic for Applicatons (VBA) 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 Outlook.NameSpace
    Dim objViews As Outlook.Views
    Dim objView As Outlook.View
    Dim strTypes As String
    Set olApp = New 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:" & vbCr & _
        vbCr & 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.

expression.Visible

*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.

This Microsoft Visual Basic/Visual Basic for Applications (VBA) example toggles the visible state of the <u>Shortcuts pane</u>.

```
Sub ShowHideShortcutsBar()
    Dim myOlApp As New Outlook.Application
    Dim myOlBar As Outlook.OutlookBarPane
    Set myOlBar = myOlApp.ActiveExplorer.Panes.Item("OutlookBar"
    myOlBar.Visible = Not myOlBar.Visible
End Sub
```

If you use Microsoft Visual Basic Scripting Edition (VBScript) in a Microsoft Outlook form, you do not create the <u>Application</u> object. This example shows how to perform the same task using VBScript code.

```
Sub CommandButton1_Click()
Set myOlBar = Application.ActiveExplorer.Panes.Item("OutlookBar")
myOlBar.Visible = Not myOlBar.Visible
End Sub
```

# **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 <u>VotingOptions</u> 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 <u>URL</u> of the Web page for the contact. Read/write.

expression.WebPage

*expression* Required. An expression that returns a <u>ContactItem</u> object.



## WebViewOn Property

Returns or sets a **Boolean** indicating the Web <u>view</u> state for a folder. **True** to display the Web page specified by the <u>WebViewURL</u> property. Read/write.

expression.WebViewOn

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

### Remarks

This property is always **False** if the value of the **WebViewURL** property is empty.

Also, setting the **WebViewOn** property to **True** before setting the **WebViewURL** property will not display the home page specified in the **WebViewURL** property.

The following Visual Basic for Applications (VBA) example creates a subfolder under the **Inbox** folder and assigns a home page to it.

```
Sub SetupFolderHomePage()
   Dim outApp As New Outlook.Application
   Dim nsp As Outlook.NameSpace
   Dim mpfInbox As Outlook.MAPIFolder
   Dim mpfNew As Outlook.MAPIFolder
   Set nsp = outApp.GetNamespace("MAPI")
   Set mpfInbox = nsp.GetDefaultFolder(olFolderInbox)
   Set mpfNew = mpfInbox.Folders.Add("MyFolderHomePage")
   mpfNew.WebViewURL = "http://www.microsoft.com"
   mpfNew.WebViewOn = True
End Sub
```



## WebViewURL Property

Returns or sets a **String** indicating the <u>URL</u> of the Web page that is assigned to a folder. Read/write.

#### expression.WebViewURL

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

The following Visual Basic for Applications (VBA) example creates a subfolder under the **Inbox** folder and assigns a home page to it.

```
Sub SetupFolderHomePage()
   Dim outApp As New Outlook.Application
   Dim nsp As Outlook.NameSpace
   Dim mpfInbox As Outlook.MAPIFolder
   Dim mpfNew As Outlook.MAPIFolder
   Set nsp = outApp.GetNamespace("MAPI")
   Set mpfInbox = nsp.GetDefaultFolder(olFolderInbox)
   Set mpfNew = mpfInbox.Folders.Add("MyFolderHomePage")
   mpfNew.WebViewURL = "http://www.microsoft.com"
   mpfNew.WebViewOn = True
End Sub
```

# 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.WindowState

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

This Microsoft Visual Basic/Visual Basic for Applications example minimizes all open explorer windows. It uses the <u>Count</u> property and <u>Item</u> method of the <u>Explorers</u> collection to enumerate the open explorer windows.

```
Sub MinimizeWindows()
    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
End Sub
```

Ena Sub

If you use VBScript, you do not create the <u>Application</u> object, and you cannot use named constants. This example shows how to use the WindowState property using VBScript.

```
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.

### Remarks

Microsoft Outlook blocks code that attempts to access the **WordEditor** property for security reasons. If you run a third-party add-in, custom solution, or other program that uses the **WordEditor** property in Office Outlook 2003, you may receive the following warning:

A program is trying to access e-mail addresses you have stored in Outlook. Do you want to allow this? If this is unexpected, it may be a virus and you should choose "No".

The following Microsoft Visual Basic Scripting Edition (VBScript) example shows how to add 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 <u>View</u> object.

#### Remarks

To determine how the XML should be structured when creating views, you can create a view by using the Outlook user interface and then you can retrieve the XML property for that view.

The following Microsoft Visual Basic/Visual Basic for Applications (VBA) example creates an instance of the **Views** collection and displays the XML definition of a view called "Table View". If the view does not exist, it creates one.

```
Sub DisplayViewDef()
'Displays the XML definition of a View object
    Dim olApp As Outlook.Application
    Dim objName As Outlook.NameSpace
    Dim objViews As Outlook.Views
    Dim objView As Outlook.View
    Set olApp = Outlook.Application
    Set objName = olApp.GetNamespace("MAPI")
    Set objViews = objName.GetDefaultFolder(olFolderInbox).Views
    'Return a view called Table View if it already exists, else crea
    Set objView = objViews.Item("Table View")
    If objView Is Nothing Then
          Set objView = objViews.Add("Table View", olTableView, olVi
    End If
    MsgBox objView.XML
End Sub
```

Following are the modified properties that 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> <bitmap>1</bitmap> <style>padding-left:3px;text-align:center;padding-left:3px</style> <displayformat>3</displayformat> </column>

# 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 <u>ContactItem</u> 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 <u>ContactItem</u> 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 <u>ContactItem</u> 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.

This code example uses the <u>WindowState</u> 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.

```
Dim myOlApp As New Outlook.Application
Public WithEvents myOlExp As Outlook.Explorer
Public Sub Initialize_handler()
    Set myOlExp = myOlApp.ActiveExplorer
End Sub
Private Sub myOlExp_Activate()
    If myOlExp.WindowState = olNormalWindow Then _
    myOlExp.WindowState = 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 <u>Application</u> declared with events in a class module.

*SearchObject* The <u>Search</u> object returned by the <u>AdvancedSearch</u> method.

## Remarks

This event only fires when the **AdvancedSearch** method is executed programmatically.

The following Visual Basic for Applications (VBA) example searches the **Inbox** for items where the subject is equal to "Test" and displays the names of the senders of the e-mail items returned by the search. The AdvanceSearchComplete event procedure sets the boolean blnSearchComp to **True** when the search is complete. This boolean variable is used by the TestAdvancedSearchComplete() procedure to determine when the search is complete. The sample code must be placed in a class module such as **ThisOutlookSession**. The **TestAdvancedSearchComplete()** procedure must be called before the event procedure can be called by Microsoft Outlook.

```
Public blnSearchComp As Boolean
Private Sub Application_AdvancedSearchComplete(ByVal SearchObject As
        MsgBox "The AdvancedSearchComplete Event fired."
        blnSearchComp = True
End Sub
Sub TestAdvancedSearchComplete()
        Dim sch As Outlook.Search
        Dim rsts As Outlook.Results
        Dim i As Integer
        blnSearchComp = False
        Const strF As String = "urn:schemas:mailheader:subject = 'Te
        Const strS As String = "Inbox"
        Set sch = Application.AdvancedSearch(strS, strF)
        While blnSearchComp = False
                DoEvents
        Wend
        Set rsts = sch.Results
        For i = 1 To rsts.Count
                MsgBox rsts.Item(i).SenderName
        Next
End Sub
```

# AdvancedSearchStopped Event

Occurs when a specified <u>Search</u> object's <u>Stop</u> method has been executed.

# **Private Sub** *application*\_ **AdvancedSearchStopped**(**ByVal** *SearchObject* **As Object**)

*expression* A variable that references an object of type <u>Application</u> declared with events in a class module.

*SearchObject* The <u>Search</u> object returned by the <u>AdvancedSearch</u> method.

## Remarks

After this event is fired, the **Search** object's **<u>Results</u>** collection will no longer be updated. This event can only be triggered programmatically.

The following Visual Basic for Applications (VBA) example starts searching the **Inbox** for items with subject equal to "Test" and immediately stops the search. This causes the AdvanceSearchStopped event procedure to be run. The sample code must be placed in a class module such as **ThisOutlookSession**. The StopSearch() procedure must be called before the event procedure can be called by Microsoft Outlook.

```
Sub StopSearch()
    Dim sch As Outlook.Search
    Dim strScope As String
    Dim strFilter As String
    strScope = "Inbox"
    strFilter = "urn:schemas:httpmail:subject = 'Test'"
    Set sch = Application.AdvancedSearch(strScope, strFilter)
    sch.Stop
End Sub
Private Sub Application_AdvancedSearchStopped(ByVal SearchObject As
    'Inform the user that the search has stopped.
    MsgBox "An AdvancedSearch has been interrupted and stopped.
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 Microsoft Visual Basic Scripting Edition (VBScript) in an Outlook form, use the word **Item**.

*Attachment* Required. The <u>Attachment</u> that was added to the item.

This Visual Basic for Applications (VBA) example checks the size of the item after an attachment has been added and displays a warning if the size exceeds 500,000 bytes. The sample code must be placed in a class module such as **ThisOutlookSession**, and the TestAttachAdd() procedure should be called before the event procedure can be called by Microsoft Outlook.

```
Public WithEvents newItem As Outlook.MailItem
Private Sub newItem_AttachmentAdd(ByVal newAttachment As Attachment)
        If newAttachment.Type = olByValue Then
                newItem.Save
                If newItem.Size > 500000 Then
                        MsgBox "Warning: Item size is now " & newIte
                End If
        End If
End Sub
Public Sub TestAttachAdd()
        Dim olApp As New Outlook.Application
        Dim atts As Outlook.Attachments
        Dim newAttachment As Outlook.Attachment
        Set newItem = olApp.CreateItem(olMailItem)
        newItem.Subject = "Test attachment"
        Set atts = newItem.Attachments
        Set newAttachment = atts.Add("C:\Test.txt", olByValue)
End Sub
```

This VBScript example shows how to use the **AttachmentAdd** event in VBScript.

```
Sub Item_AttachmentAdd(ByVal newAttachment)
    If newAttachment.Type = 1 Then
        Item.Save
        If Item.Size > 500000 Then
                  MsgBox "Warning: Item size is now " & Item.S
                  End If
        End If
End Sub
```

## AttachmentRead Event

Occurs when an attachment in an e-mail 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 Microsoft Visual Basic Scripting Edition (VBScript), use the word **Item**.

*Attachment* Required. The <u>Attachment</u> that was opened.

This Visual Basic for Applications (VBA) example displays a message when the user tries to read an attachment. The sample code must be placed in a class module such as **ThisOutlookSession**, and the TestAttachRead() procedure should be called before the event procedure can be called by Microsoft Outlook. For this example to run, there has to be at least one item in the Inbox with subject as 'Test' and containing at least one attachment.

This VBScript example reminds the user to also save changes to the original file.

## **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 Microsoft Visual Basic Scripting Edition (VBScript), use the word **Item**.

*Attachment* Required. The <u>Attachment</u> 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

This event corresponds to when attachments are saved to the messaging store. The **BeforeAttachmentSave** event occurs just before an attachment is saved when an item is saved. If a user edits an attachment and then saves those changes, the **BeforeAttachmentSave** event will not occur at that time; instead it will occur when the item itself is later saved. It also does not occur when the attachment is saved on the hard disk using the **SaveAsFile** method.

In VBScript, if you set the return value of this function to **False**, the save operation is cancelled and the attachment is not changed.

This Visual Basic for Applications (VBA) example notifies the user that the user is not allowed to save the attachment. The **Cancel** argument is set to **True** to cancel the save operation. The sample code must be placed in a class module such as **ThisOutlookSession**, and the TestAttachSave() procedure should be called before the event procedure can be called by Microsoft Outlook.

## **BeforeCheckNames Event**

Occurs just before Microsoft Outlook starts resolving names in the recipient collection for an e-mail item.

#### Sub object\_BeforeCheckNames(Cancel As Boolean)

*object* An expression that evaluates to an object in the Applies To list. In Microsoft Visual Basic Scripting Edition (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

You use the **BeforeCheckNames** event in VBScript, but the event does not fire when an e-mail name is resolved on the form.

The event does not fire under the following circumstances:

- You customized a Journal Entry form and then resolved a contact in the **Contacts** field.
- You customized a Contact form and then resolved a contact in the **Contacts** field.
- You customized any type of form and Outlook automatically resolved the name in the background.
- You programmatically created and resolved a recipient.

This Visual Basic for Applications (VBA) example asks the user if the user wants to resolve names and returns **False** to cancel the operation if the user answers no. The sample code must be placed in a class module such as **ThisOutlookSession**, and the SendMail() procedure should be called before the event procedure can be called by Outlook.

## **BeforeDelete Event**

Occurs before an e-mail 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 an e-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.

The following Visual Basic for Applications (VBA) example prompts the user regarding whether to delete the item currently open. For this example to run, you need to have an open e-mail item that can be deleted. If you click **No**, the item will not be deleted. If this event is canceled, Microsoft Outlook displays an error message. Therefore, you need to capture this event in your code. One way to do this is shown below. The sample code must be placed in a class module such as **ThisOutlookSession**, and the DeleteMail() procedure should be called before the event procedure can be called by Outlook.

```
Public WithEvents myItem As Outlook.MailItem
Public olApp As New Outlook.Application
Public Sub DeleteMail()
        Const strCancelEvent = "Application-defined or object-define
        On Error GoTo ErrHandler
        Set olApp = CreateObject("Outlook.Application")
        Set myItem = olApp.ActiveInspector.CurrentItem
        myItem.Delete
        Exit Sub
        ErrHandler:
                MsgBox Err.Description
                If Err.Description = strCancelEvent Then
                                MsgBox "The event was cancelled."
                        End If
                'If you want to execute the next instruction
                        Resume Next
                'Otherwise it will finish here
End Sub
Private Sub myItem_BeforeDelete(ByVal Item As Object, Cancel As Bool
        '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
        End If
End Sub
```

## **BeforeFolderSwitch Event**

Occurs before the explorer goes to a new folder, either as a result of user action or through program code. This event is not available in Microsoft Visual Basic Scripting Edition (VBScript).

**Note** If the folder being switched to is in a namespace 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 <u>MAPIFolder</u> 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.

This sample prevents a user from switching to a folder named "Off Limits". The sample code must be placed in a class module such as **ThisOutlookSession**, and the Initialize\_handler routine must be called before the event procedure can be called by Microsoft Outlook. To run this example without errors, make sure a folder by the name 'Off Limits' exists in the folder displayed in the active explorer.

```
Dim myOlApp As New 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, Ca
   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 <u>Shortcuts pane</u>, either as a result of user action or through program code. This event is not available in Microsoft Visual Basic Scripting Edition (VBScript).

#### Sub *object\_*BeforeGroupAdd(Cancel As Boolean)

*object* An expression that evaluates to an <u>**OutlookBarGroups**</u> 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 <u>Shortcuts pane</u>.

This Visual Basic for Applications (VBA) example prevents the user from adding a group to the <u>Shortcuts pane</u>. The sample code must be placed in a class module such as **ThisOutlookSession**, and the Initialize\_handler routine must be called before the event procedure can be called by 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_BeforeGroupAdd(Cancel As Boolean)
Cancel = True
End Sub
```



## **BeforeGroupRemove Event**

Occurs before a new group is removed from the <u>Shortcuts pane</u>, either as a result of user action or through program code. This event is not available in Microsoft Visual Basic Scripting Edition (VBScript).

## Sub *object\_*BeforeGroupRemove(ByVal Group As OutlookBarGroup, Cancel As Boolean)

*object* An expression that evaluates to an <u>**OutlookBarGroups**</u> collection object.

*Group* Required. The <u>OutlookBarGroup</u> 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 <u>Shortcuts pane</u>.

This Microsoft Visual Basic/Visual Basic for Applications (VBA) example prevents the user from removing a group from the Shortcuts pane. The sample code must be placed in a class module such as **ThisOutlookSession**, and the Initialize\_handler routine must be called before the event procedure can be called by Outlook. You will still be prompted when you try to delete a shortcut. However, the group will not be deleted even if you clicked **Yes**.

```
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 OutlookBarGr
Cancel = True
End Sub
```



# **BeforeGroupSwitch Event**

With the new <u>Navigation Pane</u> and <u>Shortcuts pane</u> in Microsoft Office Outlook 2003, this event does not fire in Office Outlook 2003.

## **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.

The following Microsoft Visual Basic/Visual Basic for Applications (VBA) 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 to the Clipboard. The sample code must be placed in a class module such as **ThisOutlookSession**, and the Initialize\_handler routine must be called before the event procedure can be called by Microsoft Outlook.

```
Public WithEvents myOlExp As Outlook.Explorer
Sub Initalize_Handler()
Set myOlExp = Application.ActiveExplorer
End Sub
Private Sub myOlExp_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?", vbYesN
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.

The following Microsoft Visual Basic/Visual Basic for Applications (VBA) 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. The sample code must be placed in a class module such as **ThisOutlookSession**, and theInitialize\_handler routine must be called before the event procedure can be called by Microsoft Outlook.

Public WithEvents myOlExp As Outlook.Explorer Sub Initalize\_Handler() Set myOlExp = Application.ActiveExplorer End Sub Private Sub myOlExp\_BeforeItemCut(Cancel As Boolean) 'Prompts the user before cutting an item Dim lngAns As Long 'Display question to user lngAns = MsgBox("Are you sure you want to cut the item?", vbYesNc 'Set cancel argument based on user's 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.

The following Microsoft Visual Basic/Visual Basic for Applications (VBA) example prompts the user before pasting the contents of the Clipboard to the specified target. If the user clicks **Yes**, the current content in the Clipboard is copied to the specified target destination. The sample code must be placed in a class module such as **ThisOutlookSession**, and the Initialize\_handler routine must be called before the event procedure can be called by Outlook.

```
Public WithEvents myOlExp As Outlook.Explorer
Sub Initalize_Handler()
Set myOlExp = Application.ActiveExplorer
End Sub
Private Sub myOlExp_BeforeItemPaste(ClipboardContent As Variant, ByV
Dim lngAns As Integer 'users' answer
'Prompt user about paste
lngAns = MsgBox("Are you sure you want to paste the contents
& Target.Name & "?", vbYesNo)
If lngAns = vbNo Then
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.

The following Microsoft Visual Basic/Visual Basic for Applications (VBA) example prompts the user with a warning message before maximizing the current window. If the user clicks **Yes**, the explorer will maximize. The sample code must be placed in a class module such as **ThisOutlookSession**, and the Initialize\_Handler() subroutine should be called before the event procedure can be called by Microsoft Outlook.

```
Public WithEvents myOlExp As Outlook.Explorer
Sub Initalize_Handler()
Set myOlExp = Application.ActiveExplorer
End Sub
Private Sub myOlExp_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 w
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)

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

The following Microsoft Visual Basic/Visual Basic for Applications (VBA) example prompts the user with a message before the window is minimized. If the user clicks **Yes**, the explorer is minimized. The sample code must be placed in a class module such as **ThisOutlookSession**, and the Initialize\_Handler() subroutine should be called before the event procedure can be called by Microsoft Outlook.

```
Public WithEvents myOlExp As Outlook.Explorer
Sub Initalize_Handler()
Set myOlExp = Application.ActiveExplorer
End Sub
Private Sub myOlExp_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 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.

The following Microsoft Visual Basic/Visual Basic for Applications (VBA) 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. The sample code must be placed in a class module such as **ThisOutlookSession**, and the Initialize\_Handler() subroutine should be called before the event procedure can be called by Microsoft Outlook.

```
Public WithEvents myOlExp As Outlook.Explorer
Sub Initalize_Handler()
Set myOlExp = Application.ActiveExplorer
End Sub
Private Sub myOlExp_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 a shortcut in the **Shortcuts** pane to navigate to a different folder. This event is not available in Microsoft Visual Basic Scripting Edition (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.

This Microsoft Visual Basic/Visual Basic for Applications (VBA) example prevents the user from using the **Shortcuts** pane 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. If you do not have a shortcut to the **Notes** folder already, you need to create one to run this example.

```
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.Outloo
   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 box is displayed.

Private Sub expression\_BeforeReminderShow(Cancel As Boolean)

*expression* A variable which references an object of type <u>**Reminders**</u> declared with events in a class module.

*Cancel* Required. **True** to cancel the event. The default value is **False**.



# **BeforeShortcutAdd Event**

Occurs before a new shortcut is added to a group in the <u>Shortcuts pane</u>, either as a result of user action or through program code. This event is not available in Microsoft Visual Basic Scripting Edition (VBScript).

#### Sub object\_BeforeShortcutAdd(Cancel As Boolean)

*object* An expression that evaluates to an <u>**OutlookBarShortcuts**</u> 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.

The following Microsoft Visual Basic/Visual Basic for Applications (VBA) example prevents a user from adding a shortcut to the first group in the <u>Shortcuts</u> pane. 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 <u>Shortcuts pane</u>, either as a result of user action or through program code. This event is not available in Microsoft Visual Basic Scripting Edition (VBScript).

## Sub *object\_*BeforeShortcutRemove(ByVal Shortcut As OutlookBarShortcut, *Cancel* As Boolean)

*object* An expression that evaluates to an <u>**OutlookBarShortcuts**</u> collection object.

*Shortcut* Required. The <u>OutlookBarShortcut</u> 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.

The following Microsoft Visual Basic/Visual Basic for Applications (VBA) example prevents a user from removing a shortcut from the Shortcuts pane. 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_BeforeShortcutRemove(ByVal Shortcut As Out
    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.

The following Microsoft Visual Basic/Visual Basic for Applications (VBA) example prompts the user with a warning message before the Inspector is sized. If the user clicks **Yes**, the inspector can be sized. The sample code must be placed in a class module such as **ThisOutlookSession**, and the Initialize\_Handler() subroutine should be called before the event procedure can be called by Microsoft Outlook.

```
Public WithEvents myIns As Outlook.Inspector
Public Sub Initalize_Handler()
    Set myIns = Application.ActiveInspector
End Sub
Private Sub myIns_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
    If lngAns = vbYes Then
        Cancel = False
    Else
        Cancel = True
    End If
End Sub
```



## **BeforeViewSwitch Event**

Occurs before the explorer changes to a new <u>view</u>, either as a result of user action or through program code. This event is not available in Microsoft Visual Basic Scripting Edition (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.

This Microsoft Visual Basic/Visual Basic for Applications (VBA) 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 New 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, Cance
   Dim Prompt As String
   Prompt = "Are you sure you want to switch to the " & NewView & "
   If MsgBox(Prompt, vbYesNo + vbQuestion) = vbNo Then Cancel = Tru
End Sub
```



## **Close Event**

Occurs when the inspector associated with a Microsoft Outlook item or when an explorer is being closed.

#### Sub object\_Close(Cancel As Boolean)

*object* An expression that evaluates to an object in the Applies To list. In Microsoft Visual Basic Scripting Edition (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 Microsoft Visual Basic Scripting Edition (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 <u>**Close**</u> method to fire this event, it can only be canceled if the **Close** method uses the *olPromptForSave* argument.

This Microsoft Visual Basic/Visual Basic for Applications (VBA) example tests for the **Close** event and if the <u>item</u> has not been <u>Saved</u>, it uses the <u>Save</u> method to save the item without prompting the user.

```
Public WithEvents myItem As Outlook.MailItem
Public Sub Initalize_Handler()
    Set myItem = Application.ActiveInspector.CurrentItem
End Sub
Private Sub myItem_Close(Cancel As Boolean)
    If Not myItem.Saved Then
        myItem.Save
        MsgBox " The item was saved."
    End If
End Sub
```



## **CustomAction Event**

Occurs when a custom action of a Microsoft Outlook <u>item</u> executes. The <u>Action</u> 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 Microsoft Visual Basic Scripting Edition (VBScript), if you set the return value of this function to **False**, the custom action operation is not completed.

This Visual Basic for Applications (VBA) example uses the <u>CustomAction</u> event to set the **Subject** property on the response item. Execute the AddAction procedure before executing the Initialize\_Handler to create an item with a custom event called 'Link Original'.

```
Public WithEvents myItem As Outlook.MailItem
Dim myOlApp As New Outlook.Application
Sub AddAction()
Dim myAction As Outlook.Action
Set myItem = myOlApp.CreateItem(olMailItem)
Set myAction = myItem.Actions.Add
myAction.Name = "Link Original"
myAction.ShowOn = olMenuAndToolbar
myAction.ReplyStyle = olLinkOriginalItem
myItem.To = "Dan Wilson"
myItem.Subject = "Before"
myItem.Send
End Sub
Sub Initialize Handler()
Set myItem = myOlApp.ActiveInspector.CurrentItem
End Sub
Private Sub myItem_CustomAction(ByVal Action As Object, ByVal Respon
Select Case Action.Name
        Case "Link Original"
            Response.Subject = "Changed by VB Script"
        Case Else
 End Select
End Sub
```



# **CustomPropertyChange Event**

Occurs when a custom property of a Microsoft Outlook <u>item</u> 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 objects in the Applies To list. In VBScript, use the word **Item**.

*Name* Required. The name of the custom property that was changed.

This Microsoft Visual Basic Scripting Edition (VBScript) example uses the <u>**CustomPropertyChange**</u> 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 <u>form</u>. 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 stops being the active window, either as a result of user action or through program code. This event is not available in Microsoft Visual Basic Scripting Edition (VBScript).

#### Sub object\_Deactivate()

*object* An expression that evaluates to an **Explorer** or **Inspector** object.

This Visual Basic for Applications (VBA) example uses the <u>WindowState</u> 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.

```
Dim myOlApp As New 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 Microsoft Visual Basic Scripting Edition (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.

The following Microsoft Visual Basic/Visual Basic for Applications (VBA) 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 Microsoft Visual Basic Scripting Edition (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.

This Microsoft Visual Basic/Visual Basic for Applications (VBA) 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).Fold
End Sub
Private Sub myFolders_FolderChange(ByVal Folder As Outlook.MAPIFolde
If Folder.Items.Count = 0 Then
MyPrompt = Folder.Name & " is empty. Do you want to delete i
If MsgBox(MyPrompt, vbYesNo + vbQuestion) = vbYes Then
Folder.Delete
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 Microsoft Visual Basic Scripting Edition (VBScript).

#### Sub object\_FolderRemove()

*object* An expression that evaluates to a **Folders** collection object.

The following Microsoft Visual Basic/Visual Basic for Applications (VBA) example displays a warning message when the user tries to a delete a folder in the Inbox. 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 myNS As Outlook.NameSpace
Dim WithEvents myFolders As Outlook.Folders
Sub Initialize_handler()
        Set myNS = myolapp.GetNamespace("MAPI")
        Set myFolders = myNS.GetDefaultFolder(olFolderInbox).Folders
End Sub
Private Sub myFolders_FolderRemove()
        MsgBox ("All the items in the folder are deleted as well.")
End Sub
```

# **FolderSwitch Event**

Occurs when the explorer goes to a new folder, either as a result of user action or through program code. This event is not available in Microsoft Visual Basic Scripting Edition (VBScript).

#### Sub object\_FolderSwitch()

*object* An expression that evaluates to an **Explorer** object.

The following Microsoft Visual Basic/Visual Basic for Applications (VBA) example displays the **Inbox** folder in "Messages" view whenever the user switches to the **Inbox** 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 myOlExp As Outlook.Explorer
Public Sub Initialize_handler()
   Set myOlExp = myOlApp.ActiveExplorer
End Sub
Private Sub myOlExp_FolderSwitch()
   Select Case myOlExp.CurrentFolder.Name
        Case "Inbox"
            myOlExp.CurrentView = "Messages"
        Case Else
   End Select
End Sub
```



# **Forward Event**

Occurs when the user selects the **Forward** action for a Microsoft Outlook <u>item</u>.

#### Sub *object\_*Forward(ByVal *Forward* As Object, *Cancel* As Boolean)

*object* An expression that evaluates to one of the objects in the Applies To list. In Microsoft Visual Basic Scripting Edition (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.

This Microsoft Visual Basic/Visual Basic for Applications (VBA) example uses the **Forward** event to disable forwarding on an <u>item</u> that has the subject "Do not forward" by setting the Cancel argument to **True** and it also displays a message that the item may not be forwarded. The sample code must be placed in a class module such as **ThisOutlookSession**, and the Initialize\_Handler() routine should be called before the event procedure can be called by Microsoft Outlook. A e-mail item must be open when you run Initialize\_Handler().

```
Public WithEvents myItem As Outlook.MailItem
Public Sub Initialize_Handler()
    Set myItem = Application.ActiveInspector.CurrentItem
End Sub
Private Sub myItem_Forward(ByVal Forward As Object, Cancel As Boolea
    If myItem.Subject = "Do not forward" Then
        MsgBox "You may not forward this message!"
        Cancel = True
    End If
End Sub
```



# **GroupAdd Event**

Occurs when a new group has been added to the <u>Shortcuts pane</u>. This event is not available in Microsoft Visual Basic Scripting Edition (VBScript).

#### Sub *object\_*GroupAdd(ByVal *NewGroup* As OutlookBarGroup)

*object* An expression that evaluates to an **OutlookBarGroups** object.

*NewGroup* Required. The <u>OutlookBarGroup</u> that was added.

This Microsoft Visual Basic/Visual Basic for Applications (VBA) example adds a shortcut to the **Calendar** whenever a 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.OutlookBar
Dim myFolder As Outlook.MAPIFolder
Set myFolder = myOlApp.GetNamespace("MAPI").GetDefaultFolder(olF
NewGroup.Shortcuts.Add myFolder, "Calendar"
End Sub
```

# ItemAdd Event

Occurs when one or more items are added to the specified collection. This event does not run when a large number of items are added to the folder at once. This event is not available in Microsoft Visual Basic Scripting Edition (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.

In this Visual Basic for Applications (VBA) example, when a new contact is added to the **Contacts** folder, the contact item is attached to an e-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 New Outlook.Application
Public WithEvents myOlItems As Outlook.Items
Public Sub Initialize_handler()
    Set myOlItems = myOlApp.GetNamespace("MAPI").GetDefaultFolder(ol
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 Microsoft Visual Basic Scripting Edition (VBScript).

#### 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.

This example uses the **<u>Start</u>** property of the **<u>AppointmentItem</u>** object to determine if the appointment starts after normal business hours. If it does, and if the **<u>Sensitivity</u>** property of the **<u>AppointmentItem</u>** object is not already set to **olPrivate**, the example offers to mark the appointment as private.

```
Dim myOlApp As New Outlook.Application
Public WithEvents myOlItems As Outlook.Items
Public Sub Initialize_handler()
   Set myOlItems = myOlApp.GetNamespace("MAPI").GetDefaultFolder(ol
End Sub
Private Sub myOlItems_ItemChange(ByVal Item As Object)
Dim prompt As String
   If VBA.Format(Item.Start, "h") >= "17" And Item.Sensitivity <> c
    prompt = "Appointment occurs after hours. Mark it private?"
    If MsgBox(prompt, vbYesNo + vbQuestion) = vbYes Then
        Item.Display
   End If
End If
End Sub
```

# **ItemRemove Event**

Occurs when an item is deleted from the specified collection. This event does not run when the last item in a Personal Folders file (.pst) is deleted, or if 16 or more items are deleted at once from a .pst file, Microsoft Exchange mailbox, or an Exchange public folder. This event is not available in Microsoft Visual Basic Scripting Edition (VBScript).

#### Sub object\_ItemRemove()

*object* An expression that evaluates to one of the objects in the Applies To list.

This Microsoft Visual Basic/Visual Basic for Applications (VBA) 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 New Outlook.Application
Public WithEvents myOlItems As Outlook.Items
Public Sub Initialize_handler()
   Set myOlItems = myOlApp.GetNamespace("MAPI").GetDefaultFolder(ol
End Sub
Private Sub myOlItems_ItemRemove()
   Dim myOlMItem As Outlook.MailItem
   If MsgBox("Do you want to notify the Sales Team?", vbYesNo + vbQ
        Set myOlMItem = myOlApp.CreateItem(olMailItem)
        myOlMItem.To = "Sales Team"
        myOlMItem.Subject = "Remove Contact"
        myOlMItem.Body = "Remove the following contact from your lis
        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 Microsoft Visual Basic Scripting Edition (VBScript).

#### Sub *object\_*ItemSend(ByVal *Item* As Object, *Cancel* As Boolean)

*object* An expression that evaluates to an <u>Application</u> 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.

The following Microsoft Visual Basic/Visual Basic for Applications (VBA) example shows how to cancel the <u>ItemSend</u> event in response to user input. 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.

```
Public WithEvents myOlApp As Outlook.Application
Public Sub Initialize_handler()
    Set myOlApp = CreateObject("Outlook.Application")
End Sub
Private Sub myOlApp_ItemSend(ByVal Item As Object, Cancel As Boolean
    Dim prompt As String
    prompt = "Are you sure you want to send " & Item.Subject & "
    If MsgBox(prompt, vbYesNo + vbQuestion, "Sample") = vbNo The
        Cancel = True
    End If
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 Microsoft Visual Basic Scripting Edition (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.

This Microsoft Visual Basic/Visual Basic for Applications (VBA) 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.

```
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.Expl
   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 Microsoft Visual Basic Scripting Edition (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.

This Microsoft Visual Basic/Visual Basic for Applications (VBA) 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.

```
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.I
        Inspector.CommandBars.Item("Standard").Visible = True
        Inspector.CommandBars.Item("Formatting").Visible = True
End Sub
```

# **NewMail Event**

Occurs when one or more new e-mail messages are received in the **Inbox**. This event is not available in Microsoft Visual Basic Scripting Edition (VBScript).

Sub object\_NewMail()

*object* An expression that evaluates to an <u>Application</u> object.

# Remarks

The **NewMail** event is useful for scenarios in which you want to be notified when a new e-mail message arrives. If you want to process items that arrive in the **Inbox**, consider using the **ItemAdd** event on the collection of items in the **Inbox**. The **ItemAdd** event passes a reference to each item that is added to a folder.

This Microsoft Visual Basic/Visual Basic for Applications (VBA) example displays the **Inbox** folder when a new e-mail message 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.

```
Public 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
    Dim x As Integer
    Set myExplorers = myOlApp.Explorers
    Set myFolder = myOlApp.GetNamespace("MAPI").GetDefaultFolder(olF
    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
     On Error GoTo 0
     myFolder.Display
End Sub
```

# NewMailEx Event

Occurs when one or more new items are received in the **Inbox**. This event passes a list of entry IDs of all the items received in the **Inbox** since the last time the event was fired. This event is not available in Microsoft Visual Basic Scripting Edition (VBScript).

#### Sub object\_NewMailEx(EntryIDs as String)

*object* An expression that evaluates to an <u>Application</u> object.

*EntryIDs* A string containing entry IDs of all items received in the **Inbox** since the last time the event was fired. The entry IDs are comma-delimited. The maximum number of entry IDs in the string are limited only by the available memory on the computer.

The **NewMailEx** event will fire for all item types received in the **Inbox** such as e-mail messages, meeting requests, and task requests. The behavior will be the same as the **NewMail** event.

The **NewMailEx** event will only fire for mailboxes in Microsoft Outlook that provide notification for received message such as Microsoft Exchange Server. Also, the event will fire only if Outlook is running. In other words, it will not fire for the new items that are received in the **Inbox** when Outlook was not open. Developers who want to access these items for customers running Outlook on an Exchange server e-mail account need to implement their code on the server. However, the **NewMailEx** event will fire against Cached Exchange Mode in all settings: **Download Full Items, Download Headers**, and **Download Headers and then Full Items**.

The following Microsoft Visual Basic for Applications (VBA) example demonstrates how the **NewMailEx** event returns the entry IDs. To run the example, run the Initialize\_Handler routine. The event will fire when one or more messages are received in the **Inbox**.

```
Public WithEvents outApp As Outlook.Application
Sub Intialize_Handler()
    Set outApp = Application
End Sub
Private Sub outApp_NewMailEx(ByVal EntryIDCollection As String)
    Dim mai As Object
    Dim intInitial As Integer
    Dim intFinal As Integer
    Dim strEntryId As String
    Dim intLength As Integer
    intInitial = 1
    intLength = Len(EntryIDCollection)
    MsgBox "Collection of EntryIds: " & EntryIDCollection
    intFinal = InStr(intInitial, EntryIDCollection, ",")
    Do While intFinal <> 0
        strEntryId = Strings.Mid(EntryIDCollection, intInitial, (int
        MsgBox "EntryId: " & strEntryId
        Set mai = Application.Session.GetItemFromID(strEntryId)
        MsgBox mai.Subject
        intInitial = intFinal + 1
        intFinal = InStr(intInitial, EntryIDCollection, ",")
    Loop
    strEntryId = Strings.Mid(EntryIDCollection, intInitial, (intLeng)
    MsqBox strEntryId
    Set mai = Application.Session.GetItemFromID(strEntryId)
    MsgBox mai.Subject
```

End Sub

### **OnError Event**

Occurs when Microsoft Outlook encounters an error while synchronizing a user's folders using the specified Send\Receive group. This event is not available in Microsoft Visual Basic Scripting Edition (VBScript).

#### Sub *object\_*OnError(ByVal Code As Long, ByVal Description As String)

*object* An expression that evaluates to a <u>SyncObject</u> object.

*Code* A unique value that identifies the error.

*Description* Required. A textual description of the error.

This Microsoft Visual Basic/Visual Basic for Applications (VBA) example displays a message box describing the synchronization error when an error occurs during 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.

```
Dim myOlApp As New Outlook.Application
Public WithEvents mySync As Outlook.SyncObject
Sub Initialize_handler()
    Set mySync = myOlApp.Session.SyncObjects.Item(1)
    mySync.Start
    mySync.Stop
End Sub
Private Sub mySync_OnError(ByVal Code As Long, ByVal Description As
    MsgBox "Unexpected sync error" & Code & ": " & Description
End Sub
```



## **Open Event**

Occurs when a Microsoft Outlook <u>item</u> is being opened in an <u>Inspector</u>. When this event occurs, the **Inspector** object is initialized but not yet displayed. The **Open** event differs from the <u>Read</u> 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 Microsoft Visual Basic Scripting Edition (VBScript), if you set the return value of this function to **False**, the open operation is not completed and the inspector is not displayed.

This Visual Basic for Applications (VBA) example uses the **Open** event to display the "All Fields" page every time the <u>item</u> is opened.

```
Public WithEvents myItem As Outlook.MailItem
Sub Initialize_handler()
    Set myItem = Application.Session.GetDefaultFolder(olFolderIn
    myItem.Display
End Sub
Private Sub myItem_Open(Cancel As Boolean)
    myItem.GetInspector.SetCurrentFormPage "All Fields"
End Sub
```

This Visual Basic for Applications 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.

```
Public WithEvents myItem As Outlook.MailItem
Sub Initialize handler()
        Set myItem = Application.Session.GetDefaultFolder(olFolderIn
        myItem.Display
End Sub
Private Sub myItem_Open(Cancel As Boolean)
        Dim mymsg As String
        If myItem.UnRead = False Then
                mymsg = "You have already read this message. Do you
                If MsgBox(mymsg, 4) = 6 Then
                        Cancel = False
                Else
                        Cancel = True
                End If
        End If
End Sub
```

## **OptionsPagesAdd Event**

Occurs whenever the **Options** dialog box (on the **Tools** menu) or a folder **Properties** dialog box is opened. This event is not available in Microsoft Visual Basic Scripting Edition (VBScript).

## Sub *object\_*OptionsPagesAdd(ByVal Pages As PropertyPages, ByVal Folder As MAPIFolder)

*object* An expression that evaluates to an <u>Application</u> or a <u>NameSpace</u> 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 <u>MAPIFolder</u> 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.

This Microsoft Visual Basic/Visual Basic for Applications (VBA) example adds a new property page to the Outlook **Options** dialog box. The sample code must be placed in a class module of a <u>Component Object Model (COM) add-in</u>.

```
Implements IDTExtensibility2
Private WithEvents OutlApp As Outlook.Application
Private Sub IDTExtensibility2_OnConnection(ByVal Application As Obje
    Set OutlApp = Application
End Sub
Private Sub OutlApp_OptionsPagesAdd(ByVal Pages As Outlook.PropertyP
    Pages.Add "PPE.SimplePage", "Simple Page"
    'PPE.SimplePage is a ProgID of the registered ActiveX Control - th
End Sub
```

### **Progress Event**

Occurs periodically while Microsoft Outlook is synchronizing a user's folders using the specified Send\Receive group. This event is not available in Microsoft Visual Basic Scripting Edition (VBScript).

## Sub *object\_Progress(ByVal State As OlSyncState, ByVal Description As String, ByVal Value As Long, ByVal Max As Long)*

*object* An expression that evaluates to a **<u>SyncObject</u>** 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.

This Microsoft Visual Basic/Visual Basic for Applications (VBA) example shows 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.

```
Dim myOlApp As New Outlook.Application
Public WithEvents mySync As Outlook.SyncObject
Sub Initialize_handler()
Set mySync = myOlApp.Session.SyncObjects.Item(1)
mySync.Start
End Sub
Private Sub mySync_Progress(ByVal State As Outlook.OlSyncState, ByVa
If Not Description = "" Then
MsgBox Description
End If
End Sub
```



### **PropertyChange Event**

Occurs when a standard property (for example, **Subject** or **To**) of a Microsoft Outlook <u>item</u> 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 Microsoft Visual Basic Scripting Edition (VBScript), use the word **Item**.

*Name* Required. The name of the property that was changed.

This Visual Basic for Applications (VBA) example uses the **PropertyChange** event to prevent someone from disabling a reminder on an <u>item</u>.

```
Public WithEvents myItem As Outlook.AppointmentItem
Sub Initialize_handler()
    Set myItem = Application.GetNamespace("MAPI").GetDefaultFold
End Sub
Private Sub myItem_PropertyChange(ByVal Name As String)
    Select Case Name
    Case "ReminderSet"
        MsgBox "You may not remove a reminder on this item."
        myItem.ReminderSet = True
    Case Else
    End Select
End Sub
```

# **Quit Event**

Occurs when Microsoft Outlook begins to close. This event is not available in Microsoft Visual Basic Scripting Edition (VBScript).

#### Sub object\_Quit()

*object* An expression that evaluates to an <u>Application</u> object.

This Microsoft Visual Basic/Visual Basic for Applications (VBA) example displays a farewell message when Microsoft Outlook exits. The sample code must be placed in a class module.

```
Private Sub Application_Quit()
    MsgBox "Goodbye, " & Application.GetNamespace("MAPI").CurrentUse
End Sub
```



## **Read Event**

Occurs when an existing Microsoft Outlook <u>item</u> 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 Microsoft Visual Basic Scripting Edition (VBScript), use the word **Item**.

This Visual Basic for Applications (VBA) example uses the **Read** event to increment a counter that tracks how often an <u>item</u> is read.

```
Public WithEvents myItem As Outlook.MailItem
Sub Initialize_handler()
    Set myItem = Application.ActiveExplorer.CurrentFolder.Items(
    myItem.Display
End Sub
Sub myItem_Read()
Dim myProperty As Outlook.UserProperty
Set myProperty = myItem.UserProperties("ReadCount")
If (myProperty Is Nothing) Then
    Set myProperty = myItem.UserProperties.Add("ReadCount", olNu
End If
myProperty.Value = myProperty.Value + 2
myItem.Save
End Sub
```



## **Reminder Event**

Occurs immediately before a <u>reminder</u> is displayed.

#### Sub *object\_*Reminder(ByVal *Item* As Object)

*object* An expression that evaluates to an <u>Application</u> object.

*Item* The <u>AppointmentItem</u>, <u>MailItem</u>, <u>ContactItem</u>, or <u>TaskItem</u> 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.

This Microsoft Visual Basic/Visual Basic for Applications (VBA) example displays the item that fired the **Reminder** event when the event fires. 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)
        Item.Display
End Sub
```

# **ReminderAdd Event**

Occurs after a reminder is added.

**Private Sub** *expression*\_**ReminderAdd**(**ByVal** *ReminderObject* **As Reminder**)

*expression* A variable which references an object of type <u>**Reminders**</u> declared with events in a class module.

*ReminderObject* Required. The <u>Reminder</u> 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.

The following example displays the date of the next reminder when a reminder is added to the collection.

```
Public WithEvents objReminders As Outlook.Reminders
Sub Initialize_handler()
    Set objReminders = Application.Reminders
End Sub
Private Sub objReminders_ReminderAdd(ByVal ReminderObject As Reminde
    'Occurs when a Reminder object is added to the collection us
    MsgBox "A new reminder is added that will fire at: " & ______
ReminderObject.NextReminderDate
```

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 <u>**Reminders**</u> declared with events in a class module.

*ReminderObject* Required. The <u>Reminder</u> object that has been modified.

The following Microsoft Visual Basic/Visual Basic for Applications (VBA) example prompts the user with a message every time a reminder is modified.

## **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 <u>**Reminders**</u> declared with events in a class module.

*ReminderObject* Required. The <u>Reminder</u> object that has been executed.

The following Microsoft Visual Basic/Visual Basic for Applications (VBA) example displays the item that fired the **Reminder** event every time a reminder is executed.

```
Public WithEvents objReminders As Outlook.Reminders
Sub Initialize_handler()
        Set objReminders = Application.Reminders
End Sub
Private Sub objReminders_ReminderFire(ByVal ReminderObject As Remind
        'Opens the item when a reminder executes
        ReminderObject.Item.Display
End Sub
```

### **ReminderRemove Event**

Occurs when a **<u>Reminder</u>** object has been removed from the collection.

#### Private Sub expression\_ReminderRemove()

*expression* A variable which references an object of type **<u>Reminders</u>** 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 **<u>Remove</u>** method.
- The **Reminder** object's **<u>Dismiss</u>** 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.

The following Microsoft Visual Basic/Visual Basic for Applications (VBA) example displays a message to the user when a <u>Reminder</u> object is removed from the collection.

```
Public WithEvents objReminders As Outlook.Reminders
Sub Initialize_handler()
    Set objReminders = Application.Reminders
End Sub
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 **<u>Reply</u>** action for a Microsoft Outlook <u>item</u>.

#### 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 Microsoft Visual Basic Scripting Edition (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.

This Visual Basic for Applications (VBA) example uses the **Reply** event and sets the **Sent Items** folder for the reply <u>item</u> to the folder in which the original item resides. To use this example, open an existing mailitem, run the Initialize Handler() procedure, then reply to the open item.

```
Public WithEvents myItem As MailItem
Sub Initialize_Handler()
    Set myItem = Application.ActiveInspector.CurrentItem
End Sub
Private Sub myItem_Reply(ByVal Response As Object, Cancel As Boolean
    Set Response.SaveSentMessageFolder = myItem.Parent
End Sub
```



# **ReplyAll Event**

Occurs when the user selects the **<u>ReplyAll</u>** action for a Microsoft Outlook <u>item</u>.

#### 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.

This Visual Basic for Applications (VBA) example uses the **ReplyAll** event and reminds the user that proceeding will reply to all original recipients of an <u>item</u> and, depending on the user's response, either allows the action to continue or stops it. To use this example, open an existing mail item, run the Initialize Handler() procedure, then reply to the item.

```
Public WithEvents myItem As MailItem
Sub Initialize_Handler()
    Set myItem = Application.ActiveInspector.CurrentItem
End Sub
Private Sub myItem_ReplyAll(ByVal Response As Object, Cancel As Bool
    Dim mymsg As String
    Dim myResult As Integer
    mymsg = "Do you really want to reply to all original recipie
    myResult = MsgBox(mymsg, vbYesNo, "Flame Protector")
    If myResult = vbNo Then
        Cancel = True
    End If
End Sub
```

# **SelectionChange Event**

Occurs when the user switches to a different item in a folder using the user interface (UI) or programmatically. This event also occurs when the user, either programmatically or using the UI, clicks or switches to a different folder that contains items as Microsoft Outlook automatically selects the first item in that folder. However, this event does not occur if the 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 Microsoft Visual Basic Scripting Edition (VBScript).

#### Sub object\_SelectionChange()

*object* An expression that evaluates to an **Explorer** object.

This Microsoft Visual Basic/Visual Basic for Applications (VBA) example displays the number of items selected in the active explorer window whenever the selection changes. 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 myOlExp As Outlook.Explorer
Public Sub Initialize_handler()
    Set myOlExp = myOlApp.ActiveExplorer
End Sub
Private Sub myOlExp_SelectionChange()
    MsgBox myOlExp.Selection.Count & " items selected."
End Sub
```



## Send Event

Occurs when the user selects the <u>Send</u> action for a Microsoft Outlook <u>item</u>.

#### 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 Microsoft Visual Basic Scripting Edition (VBScript), if you set the return value of this function to **False**, the item is not sent.

This Visual Basic for Applications (VBA) example uses the <u>Send</u> event and sends an <u>item</u> with an automatic expiration date. The sample code must be placed in a class module such as **ThisOutlookSession**, and the SendMyMail procedure must be called before the event procedure can be called by Microsoft Outlook.

```
Public WithEvents myItem As Outlook.MailItem
Sub SendMyMail()
    Set myItem = Outlook.CreateItem(olMailItem)
    myItem.To = "Dan Wilson"
    myItem.Subject = "Data files information"
    myItem.Send
End Sub
Private Sub myItem_Send(Cancel As Boolean)
    myItem.ExpiryTime = #2/2/2003 4:00:00 PM#
End Sub
```



# ShortcutAdd Event

Occurs when a new <u>shortcut</u> is added to a Shortcuts pane group. This event is not available in Microsoft Visual Basic Scripting Edition (VBScript).

#### Sub *object\_ShortcutAdd(ByVal NewShortcut As OutlookBarShortcut)*

*object* An expression that evaluates to an <u>**OutlookBarShortcuts**</u> collection object.

*NewShortcut* Required <u>OutlookBarShortcut</u> object. The shortcut that is being added.

This Microsoft Visual Basic/Visual Basic for Applications (VBA) example changes the name of a **Calendar** shortcut when it is added to the first group in the <u>Shortcuts pane</u>. 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.

## **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 **<u>Reminders</u>** declared with events in a class module.

## Remarks

This event will fire when the **<u>Snooze</u>** method is executed, or when the user clicks the **Snooze** button.

The following Microsoft Visual Basic/Visual Basic for Applications (VBA) example displays the original date and time set for the <u>Reminder</u> object that has been snoozed.

```
Public WithEvents objReminders As Outlook.Reminders
Sub Initialize_Handler()
        Set objReminders = Application.Reminders
End Sub
Private Sub objReminders_Snooze(ByVal ReminderObject As Reminder)
        'Occurs when a user clicks Snooze or when snooze is
        'programmatically executed.
        MsgBox "The reminder was originally set at " & ReminderObjec
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 Microsoft Visual Basic Scripting Edition (VBScript).

#### Sub object\_Startup()

*object* An expression that evaluates to an <u>Application</u> object.

### Remarks

An Outlook Visual Basic for Applications (VBA) macro can use this event procedure to initialize itself when Outlook starts.

This Microsoft Outlook Visual Basic for Applications example displays a welcome message to the user and maximizes the Outlook explorer window when Outlook starts.

# SyncEnd Event

Occurs immediately after Microsoft Outlook finishes synchronizing a user's folders using the specified Send\Receive group. This event is not available in Microsoft Visual Basic Scripting Edition (VBScript).

#### Sub object\_SyncEnd()

*object* An expression that evaluates to a <u>SyncObject</u> object.

This Microsoft Visual Basic/Visual Basic for Applications (VBA) example displays a message when synchronization is complete. 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
Sub Initialize_handler()
Set mySync = myOlApp.Session.SyncObjects.Item(1)
mySync.Start
End Sub
Private Sub mySync_SyncEnd()
MsgBox "Synchronization is complete."
End Sub
```

## SyncStart Event

Occurs when Microsoft Outlook begins synchronizing a user's folders using the specified Send\Receive group. This event is not available in Microsoft Visual Basic Scripting Edition (VBScript).

#### Sub object\_SyncStart()

*object* An expression that evaluates to a <u>SyncObject</u> object.

This Visual Basic for Applications (VBA) example displays a message telling the user that the synchronization might take a long time. 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
Sub Initialize_handler()
Set mySync = myOlApp.Session.SyncObjects.Item(1)
mySync.Start
End Sub
Private Sub mySync_SyncStart()
MsgBox "Synchronization is about to start. It might take a long tim
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 <u>Views</u> declared with events in a class module.

*View* The new view added to the collection prior to this event.

The following Microsoft Visual Basic/Visual Basic for Applications (VBA) example displays the view's name and saves it when the **ViewAdd** event is fired. Use the **Save** method after the properties have been modified to save the changes to the view. The sample code must be placed in a class module such as **ThisOutlookSession**, and the AddView() procedure should be called before the event procedure can be called by Microsoft Outlook.

```
Public WithEvents objViews As Outlook.Views
Sub AddView()
    Dim myOlApp As New Outlook.Application
    Dim objView As Outlook.View
    Set objViews = myOlApp.ActiveExplorer.CurrentFolder.Views
    Set objView = objViews.Add("Latest View1", olTableView, olViewSa
End Sub
Sub objViews_ViewAdd(ByVal View As View)
'Displays name of new view
With View
    Msgbox .Name & " was created programmatically."
    .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 <u>Views</u> declared with events in a class module.

*View* The view which was removed from the collection prior to this event.

The following Microsoft Visual Basic/Visual Basic for Applications (VBA) example displays the name of the view that has been removed from the collection when the **ViewRemove** event is fired. The sample code must be placed in a class module such as **ThisOutlookSession**, and the DeleteView() procedure should be called before the event procedure can be called by Microsoft Outlook.

```
Public WithEvents objViews As Outlook.Views
Sub DeleteView()
        Dim myolapp As New Outlook.Application
        Set objViews = myolapp.Application.ActiveExplorer.CurrentFol
        objViews.Item("New Table View").Delete
End Sub
Sub objViews_ViewRemove(ByVal View As View)
        'Displays view name
        MsgBox "The view: " & View.Name & " was removed programmatic
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 Microsoft Visual Basic Scripting Edition (VBScript).

#### Sub object\_ViewSwitch()

*object* An expression that evaluates to an **Explorer** object.

This Visual Basic for Applications (VBA) example hides the preview pane if it is visible when the user switches to 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 <u>Save</u> or <u>SaveAs</u> 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 Microsoft Visual Basic Scripting Edition (VBScript), if you set the return value of this function to **False**, the save operation is not completed.

This Visual Basic for Applications (VBA) example uses the <u>Write</u> event and warns the user that the <u>item</u> 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. If this event is canceled, Microsoft Outlook displays an error message. Therefore, you need to capture this event in your code. One way to do this is shown below. The sample code must be placed in a class module such as **ThisOutlookSession**, and the Initialize\_Handler() subroutine must be called before the event procedure can be called by Microsoft Outlook.

```
Public WithEvents myItem As Outlook.MailItem
Private Sub myItem_Write(Cancel As Boolean)
        Dim myResult As Integer
        myItem = "The item is about to be saved. Do you wish to over
        myResult = MsgBox(myItem, vbYesNo, "Save")
        If myResult = vbNo Then
                Cancel = True
        End If
End Sub
Public Sub Initalize Handler()
        Const strCancelEvent = "Application-defined or object-define
        On Error GoTo ErrHandler
        Set myItem = Application.ActiveInspector.CurrentItem
        mvItem.Save
        Exit Sub
        ErrHandler:
                MsgBox Err.Description
                If Err.Description = strCancelEvent Then
                        MsgBox "The event was cancelled."
                End If
End Sub
```



# **Outlook Constants**

This topic provides a list of all constants in the Outlook object model.

#### **OlActionCopyLike**

Constant	Value
olForward	2
olReply	0
olReplyAll	1
olReplyFolder	3
olRespond	4

#### **OlActionReplyStyle**

Constant	Value
olEmbedOriginalItem	1
olIncludeOriginalText	2
olIndentOriginalText	3
olLinkOriginalItem	4
olOmitOriginalText	0
olReplyTickOriginalText	1000
olUserPreference	5

**OlActionResponseStyle** 

#### **Constant Value**

olOpen 0

olPrompt 2

olSend 1

**OlActionShowOn** 

 Constant
 Value

 olDontShow
 0

 olMenu
 1

 olMenuAndToolbar
 2

#### **OlAttachmentType**

Constant	Value
olByReference	4
olByValue	1
olEmbeddeditem	5
olOLE	6

#### **OlBodyFormat**

Constant	Value
olFormatHTML	2
olFormatPlain	1
olFormatRichText	3
olFormatUnspecified	0

**OlBusyStatus** 

Constant	Value
olBusy	2
olFree	0
olOutOfOffice	3
olTentative	1

**OlConnectionMode** 

ConstantValueolLowBandwidth200olOffline100

olOnline 300

**OlDaysOfWeek** 

# Constant Value

olFriday	32
olMonday	2
olSaturday	64
olSunday	1
olThursday	16
olTuesday	4
olWednesday	8

#### **OlDefaultFolders**

Constant	Value
olFolderCalendar	9
olFolderContacts	10
olFolderDeletedItems	3
olFolderDrafts	16
olFolderInbox	6
olFolderJournal	11
olFolderJunk	23
olFolderNotes	12
olFolderOutbox	4
olFolderSentMail	5
olFolderTasks	13
ol Public Folders All Public Folders	18
olFolderConflicts	19
olFolderLocalFailures	21
olFolderServerFailures	22
olFolderSyncIssues	20

**OlDisplayType** 

Constant	Value
olAgent	3
olDistList	1
olForum	2
olOrganization	4
olPrivateDistList	5
olRemoteUser	6
olUser	0

**OlDownloadState** 

ConstantValueolFullItem1olHeaderOnly0

**OlEditorType** 

ConstantValueolEditorHTML2olEditorRTF3olEditorText1olEditorWord4

#### **OlExchangeConnectionMode**

Constant	Value
olConnected	400
olConnectedHeaders	300
olDisconnected	200
olNoExchange	0
olOffline	100
olOnline	500



Constant	Value
olBlueFlagIcon	5
olGreenFlagIcon	3
olNoFlagIcon	0
olOrangeFlagIcon	2
olPurpleFlagIcon	1
olRedFlagIcon	6
olYellowFlagIcon	4

### **OlFlagStatus**

 Constant
 Value

 olFlagComplete
 1

 olFlagMarked
 2

 olNoFlag
 0

#### <u>OlFolderDisplayMode</u>

Constant	Value
olFolderDisplayFolderOnly	1
olFolderDisplayNoNavigation	2
olFolderDisplayNormal	0

#### **OlFormRegistry**

Constant	Value
olDefaultRegistry	0
olFolderRegistry	3
olOrganizationRegistry	4
olPersonalRegistry	2

#### **OlGender**

**Constant** Value

olFemale 1 olMale 2 olUnspecified 0

#### **OlImportance**

ConstantValueolImportanceHigh2olImportanceLow0olImportanceNormal1

#### **OlInspectorClose**

Constant	Value
olDiscard	1
olPromptForSave	2
olSave	0

#### **OlItemType**

Constant	Value
olAppointmentItem	1
olContactItem	2
olDistributionListItem	7
olJournalItem	4
olMailItem	0
olNoteItem	5
olPostItem	6
olTaskItem	3

**OlJournalRecipientType** 

ConstantValueolAssociatedContact1

#### **OlMailingAddress**

#### **Constant Value**

- olBusiness 2
- olHome 1
- olNone 0
- olOther 3

#### **OlMailRecipientType**

#### Constant Value

olBCC 3 olCC 2 olOriginator 0 olTo 1

#### **OlMeetingRecipientType**

# ConstantValueolOptional2olOrganizer0olRequired1olResource3

#### **OlMeetingResponse**

ConstantValueolMeetingAccepted3olMeetingDeclined4olMeetingTentative2

**OlMeetingStatus** 

Constant Value

olMeeting 1 olMeetingCanceled 5 olMeetingReceived 3 olNonMeeting 0

#### **OlNetMeetingType**

Constant	Value
olExchangeConferencing	2
olNetMeeting	0
olNetShow	1

#### **OlNoteColor**

#### **Constant Value**

- olBlue 0
- olGreen 1
- olPink 2
- olWhite 4
- olYellow 3

#### **OlObjectClass**

Constant	Value
olAction	32
olActions	33
olAddressEntries	21
olAddressEntry	8
olAddressList	7
olAddressLists	20
olApplication	0
olAppointment	26
olAttachment	5
olAttachments	18

olConflict	117
olConflicts	118
olContact	40
olDistributionList	69
olDocument	41
olException	30
olExceptions	29
olExplorer	34
olExplorers	60
olFolder	2
olFolders	15
olFormDescription	37
olInspector	35
olInspectors	61
olItemProperties	98
olItemProperty	99
olItems	16
olJournal	42
olLink	75
olLinks	76
olMail	43
olMeetingCancellation	54
olMeetingRequest	53
olMeetingResponseNegative	55
olMeetingResponsePositive	56
olMeetingResponseTentative	57
olNamespace	1
olNote	44
olOutlookBarGroup	66
olOutlookBarGroups	65
olOutlookBarPane	63
olOutlookBarShortcut	68
olOutlookBarShortcuts	67

olOutlookBarStorage	64
olPages	36
olPanes	62
olPost	45
olPropertyPages	71
olPropertyPageSite	70
olRecipient	4
olRecipients	17
olRecurrencePattern	28
olReminder	101
olReminders	100
olRemote	47
olReport	46
olResults	78
olSearch	77
olSelection	74
olSyncObject	72
olSyncObjects	73
olTask	48
olTaskRequest	49
olTaskRequestAccept	51
olTaskRequestDecline	52
olTaskRequestUpdate	50
olUserProperties	38
olUserProperty	39
olView	80
olViews	79

## **OlOfficeDocItemsType**

ConstantValueolExcelWorkSheetItem8olPowerPointShowItem10

olWordDocumentItem 9

#### **OlOutlookBarViewType**

Constant Value olLargeIcon 0 olSmallIcon 1

#### **OlPane**

Constant	Value
olFolderList	2
olNavigationPane	4
olOutlookBar	1
olPreview	3

#### **OlPermission**

Constant	Value
olDoNotForward	1
olPermissionTemplate	2
olUnrestricted	0

**OlPermissionService** 

ConstantValueolUnknown0olWindows1olPassport2

**OlRecurrenceState** 

ConstantValueolApptException3

olApptMaster 1 olApptNotRecurring 0 olApptOccurrence 2

#### **OlRecurrenceType**

ConstantValueolRecursDaily0olRecursMonthly2olRecursWeekly1olRecursYearly5olRecursYearNth6

#### **OlRemoteStatus**

Constant	Value
olMarkedForCopy	3
olMarkedForDelete	4
olMarkedForDownload	2
olRemoteStatusNone	0
olUnMarked	1

#### **OlResponseStatus**

Constant	Value
olResponseAccepted	3
olResponseDeclined	4
olResponseNone	0
olResponseNotResponded	5
olResponseOrganized	1
olResponseTentative	2

**OlSaveAsType** 

Constant	Value
olDoc	4
olHTML	5
olICal	8
olMSG	3
olMSGUnicode	9
olRTF	1
olTemplate	2
olTXT	0
olVCal	7
olVCard	6

**OlSensitivity** 

 Constant
 Value

 olConfidential
 3

 olNormal
 0

 olPersonal
 1

 olPrivate
 2

**OlShowItemCount** 

Constant	Value
olNoItemCount	0
olShowTotalItemCount	2
olShowUnreadItemCount	1

#### <u>OlSortOrder</u>

ConstantValueolAscending1olDescending2olSortNone0

#### **OlStoreType**

ConstantValueolStoreANSI3olStoreDefault1olStoreUnicode2

**OlSyncState** 

ConstantValueolSyncStarted1olSyncStopped0

#### **OlTaskDelegationState**

ConstantValueolTaskDelegationAccepted2olTaskDelegationDeclined3olTaskDelegationUnknown1olTaskNotDelegated0

<u>OlTaskOwnership</u>

Constant	Value
olDelegatedTask	1
olNewTask	0
olOwnTask	2

**OlTaskRecipientType** 

ConstantValueolFinalStatus3olUpdate2

#### **OlTaskResponse**

#### Constant Value

olTaskAccept 2 olTaskAssign 1 olTaskDecline 3 olTaskSimple 0

#### <u>OlTaskStatus</u>

Constant	Value
olTaskComplete	2
olTaskDeferred	4
olTaskInProgress	1
olTaskNotStarted	0
olTaskWaiting	3

#### **OlTrackingStatus**

Constant	Value
olTrackingDelivered	1
olTrackingNone	0
olTrackingNotDelivered	2
olTrackingNotRead	3
olTrackingRead	6
olTrackingRecallFailure	4
olTrackingRecallSuccess	5
olTrackingReplied	7

#### **OlUserPropertyType**

Constant	Value
olCombination	19
olCurrency	14

olDateTime	5
olDuration	7
olFormula	18
olKeywords	11
olNumber	3
olOutlookInternal	0
olPercent	12
olText	1
olYesNo	6

#### **OlViewSaveOption**

# Constant Value

olViewSaveOptionAllFoldersOfType 2 olViewSaveOptionThisFolderEveryone 0 olViewSaveOptionThisFolderOnlyMe 1

#### **OlViewType**

ConstantValueolCalendarView2olCardView1olIconView3olTableView0olTimelineView4

#### **OlWindowState**

ConstantValueolMaximized0olMinimized1olNormalWindow2

# **EnableSharedAttachments Property**

Sets or returns a **Boolean** that determines whether the **Attachment Options** task pane will be displayed in the Microsoft Outlook user interface for an e-mail item. Read/write.

#### expression. EnableSharedAttachments

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

# Example

The following Microsoft Visual Basic for Applications (VBA) example hides the **Attachment Options** task pane when an attachment is added to an e-mail item. The sample code must be placed in a class module such as ThisOutlookSession, and the TestAttachRead() procedure should be called before the event procedure can be called by Outlook. For this example to run without errors, a file called Test.txt should exist in the C:\ folder.

```
Public WithEvents newItem As Outlook.MailItem
Private Sub newItem_AttachmentAdd(ByVal newAttachment As Attachment)
    newItem.EnableSharedAttachments = False
    newItem.Display
End Sub
Public Sub TestAttachAdd()
    Dim olApp As New Outlook.Application
    Dim atts As Outlook.Attachments
    Dim newAttachment As Outlook.Attachment
    Set newItem = olApp.CreateItem(olMailItem)
    newItem.Subject = "Test attachment"
    Set atts = newItem.Attachments
    Set newAttachment = atts.Add("C:\Test.txt", olByValue)
End Sub
```

# **HasCoverSheet Property**

Sets or returns a **Boolean** that determines the setting of the **Use Cover Sheet** option in the Fax UI, which in turn controls what is displayed in the body of the mail item. Read/write.

#### expression.HasCoverSheet

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

# **IsIPFax Property**

Sets or returns a **Boolean** that determines if a mail item is a fax. Read/write.

#### expression.IsIPFax

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

# **Outlook Object Model**

**Application object** -<u>NameSpace object</u> | <u>PropertyPages collection</u> -<u>PropertyPage object</u> SyncObjects collection -<u>SyncObject object</u> AddressLists collection -AddressList object AddressEntries collection AddressEntry object Folders collection MAPIFolder object Items collection └<u>Item object</u> <u>Links collection</u> └<u>Link object</u> UserProperties collection L<u>UserProperty object</u> -FormDescription object Actions collection Action object Attachments collection Attachment object -Recipients collection -Recipient object -RecurrencePattern object **Exceptions** collection

-Assistant object COMAddIns collection COMAddIn object <u>Explorers collection</u> Explorer object Selection collection <u>-Items object</u> MAPIFolder object Views collection View object CommandBars collection CommandBar object -Panes collection  $^{L}$ Pane object OutlookBarPane object <u>OutlookBarStorage ol</u> <u>OutlookBarGroups</u> <u>OutlookBarGrour</u> OutlookBarShort( -OutlookBarSho Inspectors collection

Exception object
PropertyPages collection
PropertyPage object
LanguageSettings object
AnswerWizard object
Search object
Results object
Reminders collection
Reminder object

#### Legend

Collection Object Inspector object WordEditor object HTMLEditor object Pages collection Page object CommandBars collection CommandBar object Item object ItemProperties collectio ItemProperty object Conflicts collection

# **Using events with Automation**

To create an event handler for Microsoft Outlook objects in Microsoft Visual Basic or Microsoft Visual Basic for Applications (VBA) 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 <u>Application</u> 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 <u>ItemSend</u> 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.

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.

# **Outlook COM add-in template**

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 Obje
' 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 connecte
 It can be
      ext cm AfterStartup
                              Add-in was connected after Outlook sta
                              or the Connect property of the corresp
                              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 c
      instance of the add-in itself.
' The custom argument is ignored.
End Sub
Private Sub IDTExtensibility2_OnDisconnection(ByVal RemoveMode As Ad
' Occurs when the COM add-in is disconnected.
' The RemoveMode argument specifies how the COM add-in was disconnec
ı.
 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 i
ı.
                               COM Add-ins dialog box, or the Connec
```

' 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

- <u>AppointmentItem</u>
- <u>ContactItem</u>
- <u>DistListItem</u>
- <u>DocumentItem</u>
- <u>JournalItem</u>
- <u>MailItem</u>
- <u>MeetingItem</u>
- <u>NoteItem</u>
- <u>PostItem</u>
- <u>RemoteItem</u>
- <u>ReportItem</u>
- <u>TaskItem</u>
- <u>TaskRequestAcceptItem</u>
- TaskRequestDeclineItem
- <u>TaskRequestItem</u>
- <u>TaskRequestUpdateItem</u>

# **GetNameSpace Method**

Returns a **<u>NameSpace</u>** object of the specified type.

#### expression.GetNameSpace(Type)

*expression* Required. An expression that returns an <u>Application</u> object.

*Type* Required **String**. The type of name space to return.

# Remarks

The only supported name space type is "MAPI". The **GetNameSpace** method is functionally equivalent to the **Session** property, which was introduced in Microsoft Outlook 98.

## Example

This Visual Basic for Applications (VBA) example uses the <u>CurrentFolder</u> property to change the displayed folder to the user's **Calendar** folder.

```
Sub ChangeCurrentFolder()
    Dim myolApp As Outlook.Application
    Dim myNamespace As Outlook.NameSpace
    Set myolApp = CreateObject("Outlook.Application")
    Set myNamespace = myolApp.GetNamespace("MAPI")
    Set myolApp.ActiveExplorer.CurrentFolder = ______
myNamespace.GetDefaultFolder(olFolderCalendar)
End Sub
```

End Sub

If you use Microsoft Visual Basic Scripting Edition (VBScript) in an Outlook form, you do not create the <u>Application</u> object, and you cannot use named constants. This example shows how to perform the same task using VBScript code.

```
Sub CommandButton1_Click()
Set myNameSpace = Application.GetNameSpace("MAPI")
Set Application.ActiveExplorer.CurrentFolder = _
myNameSpace.GetDefaultFolder(9)
End Sub
```

# **OLE Programmatic Identifiers**

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.

To create this control	Use this identifier
CheckBox	Forms.CheckBox.1
ComboBox	Forms.ComboBox.1
CommandButton	Forms.CommandButton.1
Frame	Forms.Frame.1
Image	Forms.Image.1
Label	Forms.Label.1
ListBox	Forms.ListBox.1
MultiPage	Forms.MultiPage.1
OptionButton	Forms.OptionButton.1
ScrollBar	Forms.ScrollBar.1
SpinButton	Forms.SpinButton.1
TabStrip	Forms.TabStrip.1
TextBox	Forms.TextBox.1
ToggleButton	Forms.ToggleButton.1

## **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.

To create this object	Use one of these identifiers
Application	Access.Application
CurrentData	Access.CodeData, Access.CurrentData
CurrentProject	Access.CodeProject, Access.CurrentProject
<b>DefaultWebOptions</b> Access.DefaultWebOptions	

## **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.

	Use one of these identifiers	Comments
Application	Excel.Application	
Workbook	Excel.AddIn	
Workbook	Excel.Chart	Returns a workbook containing two worksheets; one for the chart and one for its data. The chart worksheet is the active worksheet.
Workbook	Excel.Sheet	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.

To create this object Use one of these identifiers

ApplicationMSGraph.ApplicationChartMSGraph.Chart

## **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.

To create this object	Use one of these identifiers
ChartSpace	OWC.Chart
DataSourceControl	OWC.DataSourceControl
ExpandControl	OWC.ExpandControl
PivotTable	OWC.PivotTable
RecordNavigationControl	lOWC.RecordNavigationControl
Spreadsheet	OWC.Spreadsheet

# **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.

#### To create this object Use one of these identifiers

Application Outlook.Application

## **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.

#### To create this object Use one of these identifiers

Application PowerPoint.Application

# **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.

To create this object	Use one of these identifiers
Application	Word.Application
Document	Word.Document, Word.Template
Global	Word.Global