New Objects

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

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

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

Objects

Objects
<u>ItemProperties</u>
<u>ItemProperty</u>
Reminder
Reminders
<u>Results</u>
Search
<u>View</u>
<u>Views</u>

New Events (Alphabetical)

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

Events

AdvancedSearchComplete

AdvancedSearchStopped

BeforeDelete

BeforeItemCopy

BeforeItemCut

BeforeItemPaste

BeforeMaximize

BeforeMinimize

BeforeMove

BeforeReminderShow

BeforeSize

 $\underline{MapiLogonComplete}$

ReminderAdd

ReminderChange

ReminderFire

ReminderRemove

Snooze

ViewAdd

ViewRemove

		_

New Events (by Object)

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

Objects	Events
	AdvancedSearchComplete
Application	AdvancedSearchStopped
	MapiLogonComplete
<u>AppointmentItem</u>	BeforeDelete
ContactItem	BeforeDelete
<u>DistListItem</u>	BeforeDelete
DocumentItem	BeforeDelete
	BeforeItemCopy
	BeforeItemCut
	BeforeItemPaste
Explorer	BeforeMaximize
	BeforeMinimize
	BeforeMove
	BeforeSize
	BeforeMaximize
Inchestor	BeforeMinimize
Inspector	BeforeMove
	BeforeSize
<u>JournalItem</u>	BeforeDelete
MailItem	BeforeDelete
MeetingItem	BeforeDelete
PostItem	BeforeDelete
	BeforeReminderShow
	ReminderAdd
	ReminderChange
Reminders	ReminderFire

	ReminderRemove
	<u>Snooze</u>
RemoteItem	<u>BeforeDelete</u>
<u>ReportItem</u>	<u>BeforeDelete</u>
TaskItem	<u>BeforeDelete</u>
TaskRequestAcceptItem	<u>BeforeDelete</u>
TaskRequestDeclineItem	<u>BeforeDelete</u>
<u>TaskRequestItem</u>	<u>BeforeDelete</u>
<u>TaskRequestUpdateItem</u>	<u>BeforeDelete</u>
¥7°	<u>ViewAdd</u>
<u>Views</u>	<u>ViewRemove</u>

New Methods (Alphabetical)

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

Methods AddMember AddToFavorites AddToPFFavorites AdvancedSearch CopyFile Dial Dismiss GoToDate IsSearchSynchronous RemoveMember RemoveStore Reset SetIcon ShowCategoriesDialog

New Methods (by Object)

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

Objects	Methods
	AdvancedSearch
Application	CopyFile
	<u>IsSearchSynchronous</u>
<u>AppointmentItem</u>	ShowCategoriesDialog
ContactItem	ShowCategoriesDialog
	AddMember
<u>DistListItem</u>	<u>RemoveMember</u>
	ShowCategoriesDialog
DocumentItem	ShowCategoriesDialog
<u>JournalItem</u>	ShowCategoriesDialog
MailItem	ShowCategoriesDialog
MAPIFolder	AddToFavorites
WATTFOIGE	AddToPFFavorites
MeetingItem	ShowCategoriesDialog
NameSpace NameSpace	<u>Dial</u>
Numespace	RemoveStore
OutlookBarShortcut	<u>SetIcon</u>
PostItem	ShowCategoriesDialog
Reminder	<u>Dismiss</u>
RemoteItem	ShowCategoriesDialog
<u>ReportItem</u>	ShowCategoriesDialog
TaskItem	ShowCategoriesDialog
TaskRequestAcceptItem	ShowCategoriesDialog
TaskRequestDeclineItem	ShowCategoriesDialog

ShowCategoriesDialog

TaskRequestItem

TaskRequestUpdateItem	ShowCategoriesDialog
¥7°	GoToDate
<u>View</u>	Reset

New Properties (Alphabetical)

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

Properties

AddressBookName

AppFolders

BodyFormat

CustomViewsOnly

DownloadState

Filter

FolderPath

HTMLDocument

IMAddress

InAppFolderSyncObject

InternetCodepage

IsConflict

IsSynchronous

IsUserProperty

IsVisible

ItemProperties

LastFirstNoSpaceAndSuffix

LockUserChanges

MarkForDownload

NextReminderDate

Offline

OriginalReminderDate

Reminders

Results

SaveOption
<u>Scope</u>
<u>SearchSubFolders</u>
ShowAsOutlookAB
<u>Tag</u>
<u>Views</u>
<u>XML</u>

New Properties (by Object)

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

Objects	Properties
Application	Reminders
	DownloadState
	<u>InternetCodepage</u>
<u>AppointmentItem</u>	IsConflict
	ItemProperties
	MarkForDownload
	DownloadState
	IMAddress
ContactItem	IsConflict
Contactitem	ItemProperties
	LastFirstNoSpaceAndSuffix
	MarkForDownload
	DownloadState
DistListItem	IsConflict
DistListiteiii	ItemProperties
	MarkForDownload
	DownloadState
DocumentItem	IsConflict
<u>Documentitem</u>	ItemProperties
	MarkForDownload
Exception	ItemProperties
T 1	HTMLDocument
Explorer	Views
<u>ItemProperty</u>	<u>IsUserProperty</u>
	DownloadState
	<u>IsConflict</u>
<u>JournalItem</u>	ItemProperties

MarkForDownload

BodyFormat DownloadState

InternetCodepage

MailItem IsConflict

ItemProperties

MarkForDownload

<u>AddressBookName</u>

CustomViewsOnly

MAPIFolder <u>FolderPath</u>

InAppFolderSyncObject

ShowAsOutlookAB

Views

DownloadState

IsConflict

MeetingItem ItemProperties

MarkForDownload

NameSpace Offline

DownloadState

NoteItem <u>IsConflict</u>

ItemProperties

MarkForDownload

BodyFormat

 $\underline{\textbf{DownloadState}}$

PostItem <u>InternetCodepage</u>

IsConflict

ItemProperties

MarkForDownload

IsVisible

Reminder NextReminderDate

OriginalReminderDate

DownloadState

RemoteItem <u>IsConflict</u>

ItemProperties

MarkForDownload

DownloadState

IsConflict

<u>ReportItem</u> <u>ItemProperties</u>

MarkForDownload

Filter

IsSynchronous

Search Results
Scope

SearchSubFolders

Tag

SyncObjects AppFolders

DownloadState

InternetCodepage

<u>TaskItem</u> <u>IsConflict</u>

ItemProperties

 $\underline{MarkForDownload}$

DownloadState

TaskRequestAcceptItem IsConflict

ItemProperties

MarkForDownload

DownloadState

TaskRequestDeclineItem IsConflict

ItemProperties

MarkForDownload

DownloadState

TaskRequestItem IsConflict

ItemProperties

MarkForDownload

DownloadState

TaskRequestUpdateItem IsConflict

<u>ItemProperties</u>
MarkForDownload

<u>UserProperty</u> <u>IsUserProperty</u>

LockUserChanges

<u>View</u> <u>SaveOption</u>

XML

Using Visual Basic for Applications in Outlook

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

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

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

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

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

own Outlook Visual Basic for Applications projects.
Learn about <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

wi	ll run the macr	o when you c	lick 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 program. Unlike an Outlook form, a Visual Basic for Applications form is not used to display an Outlook item, nor can a control on a Visual Basic for Applications form be bound to an item field.

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

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

```
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.

<pre>Dim myItems As Outlook.Items Set myItems = Application.GetNamespace("MAPI").GetDefaultFolder(olfo For x = 1 To myItems.Count</pre>
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 **Application** object, which is the highest-level object in the Outlook object model. All automation code must first define an Outlook **Application** object to be able to access any other Outlook objects.

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

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

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

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

```
Set objOL = New Outlook.Application
```

```
Set objNS = objOL.GetNameSpace("MAPI")
Set objFolder = objNS.GetDefaultFolder(olFolderContacts)
```

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

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

Automating Outlook from other Microsoft Office applications

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

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

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

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

object. You can also use the New keyword with the Dim , Private , Public , or Static statement to declare an object variable. The new instance of the Application object is then created on the first reference to the variable.
Automating Outlook from a Visual Basic Application provides examples of using these methods of referencing the Outlook Application object.

Working with Outlook events

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

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

Types of Events

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

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

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

Responding to Events

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

```
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.	ž

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"

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.
8	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.
16	Connect first time. The COM add-in is loaded and connected the first time the user runs Outlook after the COM add-in has been registered. The next time Outlook is run, the COM add-in is loaded when the user requests it. Use this value if your COM add-in modifies the user interface to allow the user to request the COM add-in be connected on demand (by clicking a button, for

[&]quot;Description"="Sample Outlook Add-In"

[&]quot;LoadBehavior"=dword:00000008

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.

Adding custom property pages

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

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

Create the page as an ActiveX control

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

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

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

Implement the PropertyPage object

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

Implements Outlook.PropertyPage

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

Procedure Description

Dirty property Called by Outlook in response to the **OnStatusChange** method to determine whether the user has changed a value on the property page. **Apply** method Called by Outlook to notify your program that the user has clicked the **OK** or the **Apply** button. Usually this procedure applies any property values changed by the user in the property page. **GetPageInfo** method Called by Outlook to obtain the Help file and topic associated with the property page.

Write a procedure that sets the Dirty property and calls the OnStatusChange method

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

Create a COM add-in containing an event procedure for the OptionsPagesAdd event

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

Action Object

Actions LAction NameSpace

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

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

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

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

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

Actions Object

Multiple objects

Actions

Multiple objects

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 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

AddressEntries

Multiple objects

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

Using the AddressEntries Object

The following example	le sets a referenc	e to an Address l	E ntries object.

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

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

Set myAddressEntry = myAddressList.AddressEntries(index)	

AddressEntry Object

Multiple objects

AddressEntry

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 (<i>index</i>), where <i>index</i> 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.
<pre>Set myAddressEntry = myAddressEntries.Item(index)</pre>

AddressList Object

AddressLists LaddressList Multiple objects

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

Set myAddressList = Application.Session.AddressLists("Personal Addre

Using the AddressList Object

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

AddressLists Object

NameSpace LAddressLists
Multiple objects

The **AddressLists** object contains a set of <u>AddressList</u> 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	<pre>myAddressLists = myNameSpace.AddressLists</pre>

Application Object

<u>Application</u> LMultiple objects

Represents the entire Microsoft Outlook application. This is the only object in the hierarchy that can be returned by using the **CreateObject** method or the intrinsic Visual Basic **GetObject** function.

The Outlook **Application** object has several purposes:

- As the root object, it allows access to other objects in the Outlook hierarchy.
- It allows direct access to a new <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.

The following Visual Basic for Applications 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

Multiple objects LappointmentItem
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 **CreateItem** method to create an **AppointmentItem** object that represents a new appointment.

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

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

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

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

Attachment Object

Attachments Lattachment NameSpace

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

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

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

Attachments Object

```
Multiple objects

Attachments

Multiple objects
```

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

Using the Attachments Object

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

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

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

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

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

ContactItem Object

Multiple objects ContactItem

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

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

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

The following VBScript example returns a new contact.

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

Use <u>Items</u>(*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.

<u> </u>		

DistListItem Object

Application 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 example creates and displays a new distribution list.

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

Use <u>Items</u>(*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

Application DocumentItem 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.

Exception Object

Exceptions Exception

Multiple objects

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

Note The **Exceptions** object is on the **RecurrencePattern**, not the **AppointmentItem** object itself.

Using the Exception Object

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

In Visual Basic for Applications:

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

In VBScript:

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

Exceptions Object

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

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

Using the Exceptions Object

The following example sets a reference to the Exceptions object.							
Set myExceptions = myRecurrencePattern.Exceptions							

Explorer Object

Multiple objects

Explorer

Multiple objects

Represents the window in which the contents of a folder are displayed.

Using the Explorer Object

- Use the <u>Item</u> method of the <u>Explorers</u> 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 **GetExplorer** method to return the **Explorer** object associated with a folder.

 Use the <u>Display</u> method of a <u>MAPIFolder</u> object to display a folder in its associated explorer.

Explorers Object

Application Explorers

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 or Microsoft Visual Basic for Applications.

Dim myOlApp as New Outlook.Application Set myExplorers = myOLApp.Explorers
The following example shows how to retrieve the Explorers object in VBScript
Set myExplorers= Application.Explorers

Folders Object

Multiple objects

Line Multiple objects

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

Using the Folders Object

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

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

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

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

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

```
Set myNewFolder = myFolder.Folders(1)
```

-		

▼ Show All

FormDescription Object

```
Multiple objects FormDescription
NameSpace
```

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

associated with an Outlook item.	

▼ Show All

Inspector Object

Multiple objects Linspector
Multiple objects

Represents the window in which an Outlook item is displayed.

Using the Inspector Object

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

The Inspector Object and Microsoft Word as the User's E-mail Editor					
If the user's default e-mail editor is Microsoft Word, and if the message format of the item being edited is plain text or HTML, then no Inspector object is associated with the item.					

Inspectors Object

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

Using the Inspectors Object

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

Dim myOlApp as New Outlook.Application Set myInspectors = myOLApp.Inspectors	
The following example shows how to retrieve the Inspectors object in VBScript.	
Set myInspectors= Application.Inspectors	

ItemProperties Collection

Multiple objects LtemProperties

Multiple objects

A collection of all properties associated with the item.

Using the ItemProperties collection

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

Use the <u>Add</u> method to add a new item property to the <u>ItemProperties</u> collection. Use the <u>Remove</u> method to remove an item property from the <u>ItemProperties</u> collection.

End Sub

Note You can only add or remove custom properties. Custom properties are denoted by the <u>IsUserProperty</u>.

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

ItemProperty Object

<u>ItemProperties</u> <u>ItemProperty</u> <u>NameSpace</u>

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

Using the ItemProperty object

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

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

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

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

End Sub
```

Items Object

MAPIFolder Litems
LNameSpace

An object containing **Outlook item objects** in a folder.

Using the Items Object

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

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

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

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

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

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

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

JournalItem Object

```
Multiple objects

LaurnalItem
Multiple objects
```

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

Using the JournalItem Object

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

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

Use <u>Items</u>(*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
NameSpace
```

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

Set	myLink	=	myLinks.Item("Microsoft Co	rporation")

Links Object

```
Multiple objects

Links

Multiple objects
```

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

MailItem Object

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 <u>Items</u>(*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
```

▼ Show All

MAPIFolder Object

Multiple objects LMAPIFolder
Multiple objects

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 <u>Folders</u>(*index*), which returns a folder within a name space or another folder, and the <u>Parent</u> property, which returns the containing object.

Using the MAPIFolder Object

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

There is a set of folders within an Outlook data store that support the default functionality of Outlook. Use <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, and olFolderTasks.

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

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

▼ Show All

MeetingItem Object

Multiple objects Leading Leadi

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 <u>MeetingItem</u>.

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.

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

NameSpace Object

Multiple objects

NameSpace

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

Multiple objects

NoteItem

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

▼ Show All

OutlookBarGroup Object

Multiple objects
CoutlookBarGroup
Multiple objects

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

Using the OutlookBarGroup Object

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

exa	impre:
Set	<pre>myOlBarGroup = myOutlookBarGroups.Item("Other Shortcuts")</pre>

▼ Show All

OutlookBarGroups Object

OutlookBarStorage OutlookBarGroups

Multiple objects

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

Using the OutlookBarGroups Object

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

▼ Show All

OutlookBarPane Object

Represents the Microsoft <u>Outlook Bar</u> pane in an explorer window. The Outlook Bar contains <u>shortcuts</u> to Microsoft Outlook and file-system folders arranged in groups.

Using the OutlookBarPane Object

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

	F
Set	<pre>myOlBarPane = myPanes.Item("OutlookBar")</pre>

▼ Show All

OutlookBarShortcut Object

OutlookBarShortcuts OutlookBarShortcut
NameSpace

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

Using the OutlookBarShortcut Object

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

name. For example.	
<pre>Set myOlBarShortcut = myOutlookBarShortcuts.Item("Calendar")</pre>	

▼ Show All

OutlookBarShortcuts Object

OutlookBarGroup OutlookBarShortcuts

Multiple objects

The **OutlookBarShortcuts** collection contains a set of **OutlookBarShortcut** objects representing all <u>shortcuts</u> in an <u>Outlook Bar</u> group.

Using the OutlookBarShortcuts Object

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

Set	<pre>myShortcuts = myOutlookBarGroup.Shortcuts</pre>

▼ Show All

OutlookBarStorage Object

Represents the storage for objects in the Microsoft Outlook Bar pane.

Using the OutlookBarStorage Object

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

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

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

Pages Object

Pages LNameSpace

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
NameSpace

The **Panes** object contains the panes displayed by the specified **Explorer**.

Using the Panes Object

Use the	Panes	property to	return tl	ne Panes	collection	object :	from an	Explore	ŗ
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 Outlook Bar, use the following:

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

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

PostItem Object

Application PostItem

Multiple objects

Represents a post in a public folder that others may browse. Unlike a <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 <u>MailItem</u> object, to save the **PostItem** to the target public folder instead of mailing it.

Using the PostItem Object

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

Set myItem = myOlApp.CreateItem(olPostItem)

Use Items (<i>index</i>), where <i>index</i> 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

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

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
```

PropertyPages Object

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

Using the PropertyPages Object

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

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

PropertyPageSite Object

PropertyPageSite LNameSpace

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

Lack Recipient

Multiple objects

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

Using the Recipient Object

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

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

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

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

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

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

Recipients Object

Multiple objects

Lambda Recipients

Multiple objects

The **Recipients** object contains **Recipient** objects.

Using the Recipients Object

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

Use the <u>Add</u> method to create a new **Recipient** object and add it to the **Recipients** object. The <u>Type</u> 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.

O	•	3			

▼ Show All

RecurrencePattern Object

```
Multiple objects RecurrencePattern

Multiple objects
```

Represents the pattern of incidence of recurring appointments and tasks for the associated **AppointmentItem** and **TaskItem** object.

Using the RecurrencePattern Object

Use the <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 **RecurrenceType** property. The **RecurrenceType** property is the first property you should set.

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

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

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

olRecursYearly	DayOfMonth	Thursday The Nth day of the month
	MonthOfYear	February
olRecursYearNth	Instance	The Nth Tuesday
	DayOfWeekMask	Tuesday, Wednesday, Thursday
	MonthOfYear	February

Reminder Object

Reminders Leminder
NameSpace

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

Using the Reminder object

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

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

```
Sub AddMeeting()
'Adds a new meeting and reminder to the reminders collection
Dim olApp As Outlook.Application
Dim objMeet As AppointmentItem

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

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

End S	Sub
-------	-----

Use the Reminders collection's Remove method to remove a Reminder object
from the collection. Once a reminder is removed from its associated item, the
AppointmentItem object's ReminderSet property is set to False.

Reminders Collection

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

Using the Reminders collection

End Sub

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
```

Reminders are created programmatically when a new Microsoft Outlook item is created with a reminder. For example, a reminder is created when an AppointmentItem object is created and the **AppointmentItem** object's ReminderSet property is set to **True**. Use the **AppointmentItem** object's

ReminderTime property to set the time in minutes at which the reminder will occur. The following example creates a new meeting and sets the ReminderSet property to **True**, adding a new **Reminder** object to the **Reminders** collection.

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

Dim olApp As Outlook.Application
Dim objMeet As AppointmentItem

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

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

End Sub
```

▼ Show All

RemoteItem Object

Represents a remote <u>item</u> 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.

Using the RemoteItem Object

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

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

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

The events that do not work for the **RemoteItem** object include **Onen Close**

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

Forward, Reply, ReplyAll, and Send.				
	_			

ReportItem Object

Application ReportItem

Multiple objects

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

Using the ReportItem Object

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

Results Collection

Search Results
NameSpace

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.

Search Object

Application Search

Multiple objects

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 <u>Search</u> object. The following example returns a search object named "SubjectSearch" and displays the object's <u>Tag</u> and <u>Filter</u> property values. The <u>Tag</u> property is used to identify a specific search once it has completed.

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

Private Sub Application_AdvancedSearchComplete(ByVal SearchObject As

End Sub

▼ Show All

Selection Object

Explorer Selection
NameSpace

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 Selection property to return the Selection collection from the Explorer object. For example:				
Set mySelectedItems = myExplorer.Selection				

SyncObject Object

SyncObjects SyncObject
NameSpace

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

Using the SyncObject Object

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

_	
Set	<pre>mySyncObject = mySyncObjects.Item("Daily")</pre>
	SyncObject object is read-only; you cannot change its properties or create ones.

SyncObjects Object

NameSpace SyncObjects

Multiple objects

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

Using the SyncObjects Object

Use the SyncObjects object from a NameSpace object. For example:
Set mySyncObjects = Application.GetNameSpace("MAPI").SyncObjects
The SyncObjects object is read-only. You cannot add an item to the collection.

TaskItem Object

Multiple objects

LaskItem

Multiple objects

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

Using The TaskItem Object

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

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

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

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

```
Set myItem = Application.CreateItem(3)
```

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

TaskRequestAcceptItem Object

<u>TaskRequestAcceptItem</u> 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 TaskIte object.	m

TaskRequestDeclineItem Object

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	Outloo	k o	biects,	you	cannot	create	this	object.
				,	9				,

Use the GetAssociatedTask method to return the associated with this TaskRequestDeclineItem . Vobject.	5

▼ Show All

TaskRequestItem Object

<u>TaskRequestItem</u> 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 **Assign** and **Send** methods to a **TaskItem** object to assign (delegate) the associated task to another user, the **TaskRequestItem** object is created when the item is received in the recipient's Inbox.

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

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

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

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

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

P		

TaskRequestUpdateItem Object

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 obje	ct.

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

▼ Show All

UserProperties Object

Multiple objects

Lagrange Lag

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

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.

▼ Show All

UserProperty Object

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

Represents a custom property of a Microsoft Outlook item.

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)

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

View Object

```
Multiple objects Liew
NameSpace
```

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

- The table view type (**olTableView**) allows you to view data in a simple field-based table.
- The Calendar view type (**olCalendarView**) allows you to view data in a calendar format.
- The card view type (**olCardView**) allows you to view data in a series of cards. Each card displays the information contained by the item and can be sorted.
- The icon view type (**olIconView**) allows you to view data as icons, similar to a Windows folder or explorer.
- The timeline view type (**olTimelineView**) allows you to view data as it is received in a customizable linear time line.

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

Using the View object

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

```
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.

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

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

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

End Sub

Views Collection

Multiple objects

Line Multiple objects

A collection of all **View** objects in the current folder.

Using the Views collection

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

```
Sub GetView()
'Returns a view called Table View

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

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

Use the <u>Add</u> 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.

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
```

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 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 **Application** object.

Example

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

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

▼ Show All

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.

 $expression. {\bf Active Inspector}$

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

Remarks

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

Example

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

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

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

End Sub

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

```
Sub ActiveInspector()
'Displays the name of the active Inspector

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

End Sub

ActiveWindow Method

-

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

 $expression. {\bf Active Window}$

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

Example

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

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

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

```
If TypeName(Application.ActiveWindow) = "Inspector" Then
    Application.ActiveWindow.WindowState = 1
End If
```

▼ Show All

Add Method

-

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

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

expression.Add

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.

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.

▶ <u>Add method as it applies to the **Attachments** object.</u>

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

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

expression.Add(Source, Type, Name, Address)

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

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

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

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

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

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

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

expression.Add(Folder, DisplayMode)

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 **OlFolderDisplayMode** 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**, or **olFolderTasks**. (The constants **olFolderDeletedItems**, **olFolderOutbox**, and **olFolderSentMail** cannot be specified for this argument.)

Add method as it applies to the <u>Inspectors</u> object.

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

expression.**Add**(*Item*)

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

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

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

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

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

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

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

Type Required OlUserPropertyType. The type of the new ItemProperty.

OlUserPropertyType can be one of these OlUserPropertyType constants.

olCombination

olCurrency

olDateTime

olDuration
olFormula
olKeywords
olNumber
olOutlookInternal
olPercent
olText
olYesNo

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

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

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

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

expression.Add(Type)

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

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

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

Links an item to another item by adding a <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 Outlook Bar and returns the new group as an **OutlookBarGroup** 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 Outlook Bar. Position one is at the top of the bar.

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

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

expression.Add(Target, Name, Index)

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

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

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

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

The *Target* type depends on the shortcut type. If the type is **MAPIFolder**, the shortcut represents a Microsoft Outlook folder. If the type is a **String**, the shortcut represents a file-system path or a URL.

▶ <u>Add method as it applies to the **Pages** object.</u>

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

expression.Add(Name)

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

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

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

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

expression.Add(Page, Title)

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

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

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

▶ <u>Add method as it applies to the **Recipients** object.</u>

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

expression.Add(Name)

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

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

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

Creates a new user property in the <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.

▶ <u>Add method as it applies to the **Views** object.</u>

Creates a new view. **OlViewType**.

expression.Add(Name, ViewType, SaveOption)

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

Name Required **String**. The name of the new view.

ViewType Required <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.

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

Remarks

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

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

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

▶ <u>As it applies to the **Inspectors** object.</u>

Remarks

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

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

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.

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

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

```
Set myOlApp = CreateObject("Outlook.Application")
Set myItem = myOlApp.CreateItem(olMailItem)
Set myAction = myItem.Actions.Add
```

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

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

▶ As it applies to the **AddressEntries** object.

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

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

▶ <u>As it applies to the **Links** collection</u>.

This Microsoft Visual Basic/Visual Basic for Applications example creates a new task item, and then prompts the user for the name of a contact to link to the

item. If the contact is found, it is added to the item's **Links** collection.

```
Dim myOlApp As New Outlook.Application
Dim myNameSpace As Outlook.NameSpace
Dim myFolder As Outlook.MAPIFolder
Dim myTask As Outlook.TaskItem
Dim myContact As Outlook.ContactItem
Set myTask = myOlApp.CreateItem(olTaskItem)
Set myNameSpace = myOlApp.GetNamespace("MAPI")
Set myFolder = myNameSpace.GetDefaultFolder(olFolderContacts)
tempstr = InputBox("Enter the name of the contact to link to this 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
myTask.Display
```

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

```
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
myTask.Display
```

▶ <u>As it applies to the **Folders** collection.</u>

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

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

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

VBScript.

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

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

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

▶ <u>As it applies to the **Items** collection</u>.

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

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

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

▶ <u>As it applies to the **OutlookBarGroups** collection.</u>

This Microsoft Visual Basic/Visual Basic for Applications example adds a group named Marketing as the second group in the Outlook Bar.

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

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

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

▶ As it applies to the **OutlookBarShortcuts** collection.

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

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

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

This Visual Basic for Applications example creates a new mail message, uses the **Add** method to add "Allison Klein" as a To recipient, and displays the message.

```
Set myOlApp = CreateObject("Outlook.Application")
Set myItem = myOlApp.CreateItem(olMailItem)
Set myRecipient = myItem.Recipients.Add("Allison Klein")
myItem.Display
```

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

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

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

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

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

▶ <u>As it applies to the **Views** object.</u>

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

```
Sub CreateView()
'Creates a new view
Dim olApp As Outlook.Application
```

AddMember Method

-

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

expression.AddMember(Recipient)

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

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

Remarks

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

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

```
Sub AddNewMember()
'Adds a new member to the distribution list
    Dim olApp As Outlook.Application
    Dim objItem As DistListItem
    Dim objMail As MailItem
    Dim objRcpnt As Recipient
    Set olApp = Outlook.Application
    Set objMail = olApp.CreateItem(olMailItem)
    'Create recipient for list
    Set objRcpnt = objMail.Recipients.Add("Jeff Smith")
    Set objItem = olApp.CreateItem(olDistributionListItem)
    objItem.AddMember Recipient:=objRcpnt
    'Add note to list and display
    objItem.Body = "Regional Sales Manager - NorthWest"
    objItem.Display
End Sub
```

AddMembers Method

Adds new members to a distribution list.

expression.AddMembers(Recipients)

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

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

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

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.

AddStore Method

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

expression.AddStore(Store)

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

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

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

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

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

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

AddToFavorites Method

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

expression.AddToFavorites(fNoUI, Name)

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

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

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

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

```
Sub FaveChange()
    Dim appolApp As Outlook.Application
    Dim nmsName As NameSpace 'the namespace
    Dim fldFolder As MAPIFolder 'the folder
    Dim strName As String 'user created string
    Set appolApp = Outlook.Application
    'Create instance of namespace
    Set nmsName = appolApp.GetNamespace("Mapi")
    Set fldFolder = nmsName.GetDefaultFolder(olFolderInbox)
    'Prompt user for a Favorites list name
    strName =
        InputBox("Enter the name of the folder as it will appear in
    Call FaveList(fldFolder, strName)
End Sub
Sub FaveList(ByRef fldFolder As MAPIFolder, ByVal strName As String)
'Add a Folder object to the Favorites list in Internet Explorer
    'Call method with strName as name argument
    fldFolder.AddToFavorites fNoUI:= True, Name:=strName
    'Display a message to the user
    MsgBox "The folder " & fldFolder.Name & \_
           " was added to the Internet Explorer favorites list as "
```

End Sub

AddToPFFavorites Method

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

expression. AddToPFFavorites

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

Remarks

Only Public Folders can be added to the Favorites Folder.

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

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

Dim olapp As Outlook.Application
Dim objFolder As MAPIFolder

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

End Sub
```

AdvancedSearch Method

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

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

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

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

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

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

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

Remarks

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

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

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

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

```
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
```

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.

Apply Method

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 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
```

Assign Method

-

Assigns a task and returns a <u>TaskItem</u> 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 <u>TaskRequestItem</u> object.

expression.Assign

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

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

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

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

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

$Cancel Response State\ Method$

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 car use this method to revert the task to its state before you responded.
expression.CancelResponseState
expression Required. An expression that returns a TaskItem object.

ClearConversationIndex Method

Clears the i	ndex of the conversation thread for the mail message or post.
expression.	ClearConversationIndex
expression object.	Required. An expression that returns a MailItem or PostItem

ClearRecurrencePattern Method

Removes the recurrence settings and restores the single-occurrence state for an appointment or task.			
expression.ClearRecurrencePattern			
<i>expression</i> Required. An expression that returns an AppointmentItem or TaskItem object.			

▼ Show All

Close Method

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

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 **OlInspectorClose**. The close behavior. If the item displayed within the inspector has not been changed, this argument has no effect.

OlInspectorClose can be one of these OlInspectorClose constants.

olDiscard Discard all changes without prompting.

olPromptForSave Prompt to save or discard all changes.

olSave Save all changes without prompting.

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

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

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

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

▼ Show All

Copy Method

▶ Copy method as it applies to the **View** object.

Creates a new instance of a **View** object.

expression.Copy(Name, SaveOption)

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

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

SaveOption Optional <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 example creates a copy of a view called "New Table View" and saves it in the current folder.

```
Sub CopyView()
'Copies a view

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

Set olApp = Outlook.Application
Set objViews = _
olApp.GetNamespace("MAPI").GetDefaultFolder(olFolderInbox).Views
'Create copy of View object
Set objNewView = objViews("New Table View").Copy(Name:="Table VisaveOption:=olViewSaveOptionThisFolderEveryone)
```

End Sub

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

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

```
Set myOlApp = CreateObject("Outlook.Application")
Set myNamespace = myOlApp.GetNamespace("MAPI")
Set myFolder = myNamespace.GetDefaultFolder(olFolderInbox)
Set myNewFolder = myFolder.Folders.Add("Saved Mail", olFolderDrafts)
Set myItem = myOlApp.CreateItem(olMailItem)
myItem.Subject = "Speeches"
Set myCopiedItem = myItem.Copy
myCopiedItem.Move myNewFolder
```

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

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

CopyFile Method

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

expression.CopyFile(FilePath, DestFolderPath)

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

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

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

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

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

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

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

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

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

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

▼ Show All

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 **Application** object.

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

OlItemType can be one of these OlItemType constants.

ol Appointment Item

olContactItem

olDistributionListItem

olJournalItem

olMailItem

olNoteItem

olPostItem

olTaskItem

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

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

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

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

▼ Show All

CreateItemFromTemplate Method

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

expression.CreateItemFromTemplate(TemplatePath, InFolder)

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

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

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

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

```
Set myOlApp = CreateObject("Outlook.Application")
Set myItem = myOlApp.CreateItemFromTemplate _ ("C:\Program Files myItem.Display

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

Set myItem = Application.CreateItemFromTemplate _ ("C:\Program F myItem.Display)
```

▼ Show All

CreateObject Method

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

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

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

expression.CreateObject(ObjectName)

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

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

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

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

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

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

CreateRecipient Method

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

expression.CreateRecipient(RecipientName)

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

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

This Visual Basic for Applications example uses the <u>CreateRecipient</u> method to obtain the default Calendar folder for the user Eric Lang.

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.

Delete Method

Deletes the object from the collection.

expression. Delete

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

This Visual Basic for Applications example uses the <u>GetFirst</u> method to locate the first folder in the Tasks folder and then uses the **Delete** method to delete the folder.

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

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

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

Details Method

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

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

expression.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.

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

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

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

Dial Method

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

expression.Dial(ContactItem)

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

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

The following 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 example opens the New Call dialog box with the Contact's
information.
Sub DialContact()
'Opens the New Call dialog with the contact info
    Dim olApp As Outlook.Application
    Dim objContact As ContactItem
    Set olApp = Outlook.Application
    Set objContact = olApp.GetNamespace("Mapi").
     GetDefaultFolder(olFolderContacts).Items("Jeff Smith")
    olApp.GetNamespace("Mapi").Dial objContact
End Sub
```

Dismiss Method

Dismisses the current reminder.

expression. Dismiss

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

Remarks

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

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

▼ Show All

Display Method

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

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

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

expression.Display(Modal)

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

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

This Visual Basic for Applications example uses the <u>Display</u> 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.

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

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

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

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

```
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! "
```

▼ Show All

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 **Action** object.

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

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

▼ Show All

Find Method

-

▶ Find method as it applies to the **Items** object.

Locates and returns an item.

expression.Find(Filter)

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

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

▶ Find method as it applies to the **UserProperties** object.

Locates and returns a <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 custom string that defines the search parameters.

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

```
Sub FindContact()
'Finds and displays last contacted info for a contact
    Dim olApp As Outlook.Application
    Dim objContact As ContactItem
    Dim objContacts As MAPIFolder
    Dim objNameSpace As NameSpace
    Dim objProperty As UserProperty
    Set olApp = CreateObject("Outlook.Application")
    Set objNameSpace = olApp.GetNamespace("MAPI")
    Set objContacts = objNameSpace.GetDefaultFolder(olFolderContacts
    Set objContact = objContacts.Items.Find("[FileAs] = ""Smith, 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
```

▼ Show All

FindNext Method

_

After the **Find** method runs, this method finds and returns the next Microsoft Outlook <u>item</u> 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 **Items** collection object.

This Visual Basic for Applications 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 <u>FindNext</u> methods to locate all the appointments that occur today and display them in a series of message boxes.

▼ Show All

Forward Method

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

expression.Forward

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

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

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

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

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

ForwardAsVcal Method

Forwards the AppointmentItem as a vCal; virtual calendar item. The ForwardAsVcal method returns a MailItem with the vCal file attached.				
expression.ForwardAsVcal				
<i>expression</i> 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 **Recipient** 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 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.

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

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

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

GetAssociatedAppointment Method

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

expression.GetAssociatedAppointment(AddToCalendar)

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

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

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

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

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

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

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

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

GetAssociatedTask Method

Returns a **TaskItem** object that represents the requested task.

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

expression.GetAssociatedTask(AddToTaskList)

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

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

This example accepts a <u>TaskRequestItem</u>, sending the response without displaying the inspector.

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

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

▼ Show All

GetDefaultFolder Method

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

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

expression.GetDefaultFolder(FolderType)

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

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

OlDefaultFolders can be one of these OlDefaultFolders constants.

olFolderCalendar

olFolderContacts

olFolderDeletedItems

olFolderDrafts

olFolderInbox

olFolderJournal

olFolderNotes

olFolderOutbox

olFolderSentMail

olFolderTasks

olPublicFoldersAllPublicFolders

This Visual Basic for Applications example uses the <u>GetDefaultFolder</u> method to obtain the default Calendar folder for the current user.

```
Set myOlApp = CreateObject("Outlook.Application")
Set myNameSpace = myOlApp.GetNameSpace("MAPI")
Set myCalendar = myNameSpace.GetDefaultFolder(olFolderCalendar)
```

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

```
Set myNameSpace = Application.GetNameSpace("MAPI")
Set myCalendar = myNameSpace.GetDefaultFolder(9)
```

This Visual Basic for Applications example returns the first folder in the Tasks Folders collection.

```
Set myOlApp = CreateObject("Outlook.Application")
Set myNameSpace = myOlApp.GetNameSpace("MAPI")
Set myTasks = myNameSpace.GetDefaultFolder(olFolderTasks)
Set myFolder = myTasks.Folders(1)
```

GetExplorer Method

Returns an **Explorer** object that represents a new, inactive **Explorer** object initialized with the specified folder as the current folder. This method is useful for returning a new **Explorer** object in which to display the folder, as opposed to using the **ActiveExplorer** method and setting the **CurrentFolder** property.

The **Display** method can be used to activate or show the **Explorer**.

The **GetExplorer** method takes an optional argument of an **OlFolderDisplayMode** constant.

By default, the new **Explorer** will be displayed in the **Normal** mode (**olFolderDisplayNormal**) with all interface elements displayed: a message panel on the right, an Outlook bar on the left, and a folder banner across the top from which the user can pull down a folder navigation panel. The exception to this rule is when you are calling **GetExplorer** on delegated folders which are in **No-Navigation** mode by default. You can apply more restrictions to a default mode, but you cannot lessen the restrictions by changing the **OlFolderDisplayMode**.

The explorer can also be displayed in **Folder-Only** mode (**olFolderDisplayFolderOnly**) in which the **Explorer** will display with no folder list and no Outlook bar, but the drop-down list and the navigation commands will still be available.

The most restrictive mode you can use is **No-Navigation** mode (**olFolderDisplayNoNavigation**). In this mode, the **Explorer** will display with no folder list, no drop-down folder list, and any "Go"-type menu/command bar options should be disabled. Basically, the user should not be able to navigate to any other folder within that **Explorer** window. By default, a delegated (shared) folder appears in **No-Navigation** mode.

expression.GetExplorer(DisplayMode)

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

DisplayMode Optional **Variant**. The display mode of the folder. Can be one of the following **OlFolderDisplayMode** constants: **olFolderDisplayFolderOnly**, **olFolderDisplayNoNavigation** or **olFolderDisplayNormal** (default).

This Visual Basic for Applications example uses the <u>GetExplorer</u> method to return a new, inactive <u>Explorer</u> for myFolder, and then displays it in the default mode of **olFolderDisplayNormal**.

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

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

```
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 example uses the <u>GetFirst</u> method to locate the first folder in the Tasks folder and then uses the <u>Delete</u> method to delete the folder.

```
Set myOlApp = CreateObject("Outlook.Application")
Set myNameSpace = myOlApp.GetNameSpace("MAPI")
Set myFolder = myNameSpace.GetDefaultFolder(olFolderTasks)
Set myOldFolder = myFolder.Folders.GetFirst
myOldFolder.Delete
```

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

```
Set myNameSpace = Application.GetNameSpace("MAPI")
Set myFolder = myNameSpace.GetDefaultFolder(13)
Set myOldFolder = myFolder.Folders.GetFirst
myOldFolder.Delete
```

GetFolderFromID Method

Returns a **MAPIFolder** object identified by the specified entry ID (if valid). This method is used for ease of transition between MAPI and OLE/Messaging applications and Microsoft Outlook.

expression.GetFolderFromID(EntryIDFolder, EntryIDStore)

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

EntryIDFolder Required **String**. The **EntryID** of the folder.

EntryIDStore Optional Variant. The **StoreID** for the folder.

This Visual Basic for Applications example obtains the **EntryID** and **StoreID** for a subfolder within the Tasks **Folders** collection and then calls the **GetFolderFromID** method using these values to obtain the same subfolder. The folder is then displayed.

```
Set myolapp = CreateObject("Outlook.Application")
Set myNameSpace = myolapp.GetNamespace("MAPI")
Set myFolder = myNameSpace.GetDefaultFolder(olFolderTasks)
myEntryID = myFolder.EntryID
myStoreID = myFolder.StoreID
Set myFolder = myNameSpace.GetFolderFromID(myEntryID, myStoreID)
myFolder.Display
```

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

```
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 AddressEntry object.

Start Required **Date**. Specifies the date.

MinPerChar Required **Long**. Specifies the length of each time slot in minutes. Default is 30 minutes.

CompleteFormat Optional Variant.

▼ Show All

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 **NameSpace** object.

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

EntryIDStore Optional Variant. The **StoreID** for the folder.

Remarks

EntryIDStore MAPI IDs.	e usually must	be provided v	when retrievir	ng an item base	ed on its

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

GetMember Method

Returns a **Recipient** object representing a member in a distribution list.

expression.GetMember(Index)

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

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

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

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

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

		_

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

<i>expression</i> Required. An expression that returns one of the objects in the Applies To list.

GetOccurrence Method

The **GetOccurrence** method returns a specific instance of the **AppointmentItem** object on the specified date.

Note The **GetOccurrence** method generates an error if no appointment of that series exists on the specified date.

expression.GetOccurrence(StartDate)

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

StartDate Required **Date** that represents local time.

This Visual Basic for Applications example uses <u>CreateItem</u> to create an <u>AppointmentItem</u> object. The <u>RecurrencePattern</u> is obtained for this item using the <u>GetRecurrencePattern</u> method. By setting the <u>RecurrencePattern</u> properties, <u>RecurrenceType</u>, <u>PatternStartDate</u>, and <u>PatternEndDate</u>, the appointments are now a recurring series that occur on a daily basis for the period of one year.

An **Exception** object is created when one instance of this recurring appointment is obtained using the **GetOccurrence** method and properties for this instance are altered. This exception to the series of appointments is obtained using the **GetRecurrencePattern** method to access the **Exceptions** collection associated with this series. Message boxes display the original **Subject** and **OriginalDate** for this exception to the series of appointments and the current date, time, and subject for this exception.

For a description of changes required for this example to work in VBScript, see the Note at the end of the example.

```
Public Sub cmdExample()
    Set myOlApp = New Outlook.Application
    Set myApptItem = myOlApp.CreateItem(olAppointmentItem)
   myApptItem.Start = #2/2/98 3:00:00 PM#
    myApptItem.End = #2/2/98 4:00:00 PM#
   myApptItem.Subject = "Meet with Boss"
    'Get the recurrence pattern for this appointment
    'and set it so that this is a daily appointment
    'that begins on 2/2/98 and ends on 2/2/99
     'and save it.
    Set myRecurrPatt = myApptItem.GetRecurrencePattern
   myRecurrPatt.RecurrenceType = olRecursDaily
   myRecurrPatt.PatternStartDate = #2/2/98#
    myRecurrPatt.PatternEndDate = #2/2/99#
   myApptItem.Save
    'Access the items in the Calendar folder to locate
    'the master AppointmentItem for the new series.
    Set myNamespace = myOlApp.GetNamespace("MAPI")
    Set myFolder = myNamespace.GetDefaultFolder(olFolderCalendar)
```

```
Set myItems = myFolder.Items
    Set myApptItem = myItems("Meet with Boss")
    'Get the recurrence pattern for this appointment
    'and obtain the occurrence for 3/12/98.
    myDate = #3/12/98 \ 3:00:00 \ PM#
    Set myRecurrPatt = myApptItem.GetRecurrencePattern
    Set myOddApptItem = myRecurrPatt.GetOccurrence(myDate)
    'Save the existing subject. Change the subject and
    'starting time for this particular appointment
    'and save it.
    saveSubject = myOddApptItem.Subject
    myOddApptItem.Subject = "Meet NEW Boss"
    newDate = #3/12/98 \ 3:30:00 \ PM#
    myOddApptItem.Start = newDate
    myOddApptItem.Save
    'Get the recurrence pattern for the master
    'AppointmentItem. Access the collection of
    'exceptions to the regular appointments.
    Set myRecurrPatt = myApptItem.GetRecurrencePattern
    Set myException = myRecurrPatt.Exceptions.Item(1)
    'Display the original date, time, and subject
    'for this exception.
    MsgBox myException.OriginalDate & ": " & saveSubject
    'Display the current date, time, and subject
    'for this exception.
    MsgBox myException.AppointmentItem.Start & ": " &
myException.AppointmentItem.Subject
End Sub
```

Note For this example to work properly in VBScript, there are only a few simple changes that need to be made in the code.

You don't have to retrieve the application as an object and you must use the values of the constants, so:

```
Set myOlApp = New Outlook.Application
    Set myApptItem = myOlApp.CreateItem(olAppointmentItem)
becomes:
Set myApptItem = Application.CreateItem(1)
```

```
and
myRecurrPatt.RecurrenceType = olRecursDaily
becomes:
myRecurrPatt.RecurrenceType = 0
and
Set myFolder = myNamespace.GetDefaultFolder(olFolderCalendar)
becomes:
Set myFolder = myNamespace.GetDefaultFolder(9)
```

GetPageInfo Method

Returns information about a custom property page.

expression.GetPageInfo(HelpFile, HelpContext)

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

HelpFile Required **String**. Specifies the Help file associated with the property page.

HelpContext Required **Long**. Specifies the context ID of the Help topic associated with the property page.

This Microsoft Visual Basic/Visual Basic for Applications example returns the name of the Help file and the context ID of the topic to be displayed.

Pri	HelpFil	e = "Proj	/Page_ GetF Page.chm' H_PageInf	ı ` `	elpFile A	s String,	HelpContext
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

<i>expression</i> Required. An expression that returns one of the objects in the Applies To list.

GetRecipientFromID Method

Returns a Recipient object identified by the specified entry ID (if valid). This method is used for ease of transition between MAPI and OLE/Messaging applications and Microsoft Outlook.				
expression.GetRecipientFromID(EntryID)				
<i>expression</i> Required. An expression that returns a NameSpace object.				
EntryID Required String . The EntryID of the recipient.				

GetRecurrencePattern Method

Returns a **RecurrencePattern** object that represents the recurrence attributes of an appointment or task. If there is no existing recurrence pattern, a new, empty **RecurrencePattern** object is returned.

expression. GetRecurrencePattern

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

This Visual Basic for Applications example uses <u>GetRecurrencePattern</u> to obtain the <u>RecurrencePattern</u> for an appointment.

```
Set myOlApp = CreateObject("Outlook.Application")
Set myAppt = myOlApp.CreateItem(olAppointmentItem)
Set myPattern = myAppt.GetRecurrencePattern
```

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

```
Set myAppt = Application.CreateItem(1)
Set myPattern = myAppt.GetRecurrencePattern
```

GetSharedDefaultFolder Method

Returns a **MAPIFolder** object that represents the specified default folder for the specified user. This method is most useful in the delegation scenario, where one user has delegated access to another user for one or more of their default folders (for example, their shared Calendar folder).

expression.GetSharedDefaultFolder(Recipient, FolderType)

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

Recipient Required Recipient object. The owner of the folder. The Recipient object must be resolved.

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

OlDefaultFolders can be one of these OlDefaultFolders constants.

olFolderCalendar

olFolderContacts

olFolderDeletedItems

olFolderDrafts

olFolderInbox

olFolderJournal

olFolderNotes

olFolderOutbox

olFolderSentMail

olFolderSharedRoot

olFolderTasks

olPublicFoldersAllPublicFolders

This Visual Basic for Applications example uses the <u>GetSharedDefaultFolder</u> method to resolve the <u>Recipient</u> object representing Kim Buhler, and then returns her shared default Calendar folder.

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.

GoToDate Method

Changes the date information displayed in the current viewing window.					
expression.GoToDate(Date)					
expression Required. An expression that returns a <u>View</u> object.					
Date Required Date . The date of the information you want to view.					

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 example uses **HideFormPage** to hide the "General" page of a newly-created **ContactItem** and then displays the item.

```
Set myOlApp = CreateObject("Outlook.Application")
Set myItem = myOlApp.CreateItem(olContactItem)
Set myPages = myItem.GetInspector.ModifiedFormPages
Set myPage = myPages.Add("General")
Set myInspector = myOlApp.ActiveInspector
myInspector.HideFormPage("General")
myItem.Display
```

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

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

▼ Show All

IsPaneVisible Method

_

Determines whether a specific explorer pane is visible. For example, use this method to determine if the Folder List is currently visible in the explorer. Returns **True** if the specified pane is displayed in the explorer.

Note You can also use the <u>Visible</u> property of the <u>OutlookBarPane</u> object to determine whether the <u>Outlook Bar</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 olOutlookBar olPreview

This Microsoft Visual Basic/Visual Basic for Applications sample uses the **IsPaneVisible** method to determine whether the Outlook Bar is visible and uses the **ShowPane** method to display it if it is not visible.

```
Dim myOlApp As New Outlook.Application
Set myOlExp = myOlApp.ActiveExplorer
If myOlExp.IsPaneVisible(olOutlookBar) = False Then
    myolexp.ShowPane olOutlookBar,True
End If
```

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

```
Set myOlExp = Application.ActiveExplorer
If myOlExp.IsPaneVisible(1) = False Then
    myOlExp.ShowPane 1,True
End If
```

IsSearchSynchronous Method

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

expression. Is Search Synchronous (Look In Folders)

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

LookInFolders Required **String.** The path name of the folders that the search will search through.

The	foll	owing	g examp]	le uses a	functi	on to	test	wheth	er or	not a	given	path	name
will	resi	ult in a	synchr	onous s	earch.								

ction Synch(ByVal Synch = App. Is : Function	Foldername As SearchSynchrono	 ef App A	As Application)

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.**

▼ Show All

Item Method

_

Exceptions

Returns an object from a collection. The following table shows the collections supported and the object type returned.

Exception

Collection Object Returned Actions Action AddressEntries AddressEntry AddressLists AddressList Attachments Attachment

ExplorersExplorerFoldersMAPIFolderInspectorsInspectorItemsOutlook item

Links Link

OutlookBarGroupsOutlookBarGroupOutlookBarShortcutsOutlookBarShortcut

Pages Panes Pane

PropertyPagesPropertyPageSelectionOutlook itemSyncObjectsSyncObjectRecipientsRecipientUserPropertiesUserProperties

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

All other Microsoft Outlook collections A generic **Object** representing a single object in the specified collection

expression.Item(Index)

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

Index Required **Variant**. Either the index number of the object, or a value used to match the default property of an object in the collection.

This Visual Basic for Applications example returns the second <u>MailItem</u> object in the default Inbox folder. It assumes that at least two **MailItem** objects already exist.

```
Set myOlApp = CreateObject("Outlook.Application")
Set myNameSpace = myOlApp.GetNameSpace("MAPI")
Set myFolder = _
    myNameSpace.GetDefaultFolder(olFolderInbox)
Set mySecondItem = myFolder.Items.Item(2)
```

This Visual Basic for Applications example returns the "Forward" action from the **Actions** collection.

```
Set myOlApp = CreateObject("Outlook.Application")
Set myItem = myOlApp.ActiveInspector.CurrentItem
Set myAction = myItem.Actions.Item("Forward")
```

This example returns a **MAPIFolder** object from a **Folders** collection.

```
Set myOlApp = CreateObject("Outlook.Application")
Set myNameSpace = myOlApp.GetNameSpace("MAPI")
Set myFolders = myNameSpace.Folders
Set myFolder = myFolders.Item("Public Folder")
```

This Visual Basic for Applications example also returns a **MAPIFolder** object from a **Folders** collection.

```
Set myOlApp = CreateObject("Outlook.Application")
Set myFolders = _
    myOlApp.ActiveExplorer.CurrentFolder.Folders
Set myFolder = myFolders.Item("Project X")
```

This Visual Basic for Applications example creates a contact, returns its empty Pages collection, adds three custom pages and then returns the first custom page from the collection.

```
Set myOlApp = CreateObject("Outlook.Application")
Set myItem = myOlApp.CreateItem(olContactItem)
Set myPages = myItem.GetInspector.ModifiedFormPages
myPages.Add "One"
```

```
myPages.Add "Two"
myPages.Add "Three"
Set myPage = myPages.Item("One")
```

This Visual Basic for Applications example creates a mail message, adds four **Recipient** objects, then returns the name of the third recipient from the newly-created **Recipients** collection.

```
Set myOlApp = CreateObject("Outlook.Application")
Set myItem = myOlApp.CreateItem(olMailItem)
Set myRecipients = myItem.Recipients
myRecipients.Add("Rich Andrews")
myRecipients.Add("Robin Hjellin")
myRecipients.Add("Meng Phua")
myRecipients.Add("Kim Yoshida")
msgbox myRecipients.Item(3).name
```

▼ Show All

Logoff Method

Logs the user off from the current MAPI <u>session</u> .					
expression.Logoff					
<i>expression</i> An expression that returns a NameSpace object.					

▼ Show All

Logon Method

Logs the user on to MAPI, obtaining a MAPI session.

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

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

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

Password Optional **Variant**. The password (if any), as a **String**, associated with the profile.

ShowDialog Optional **Variant**. **True** to display the MAPI logon dialog.

NewSession Optional **Variant**. **True** to create a new session (doesn't use an existing session). Multiple sessions cannot be created in Outlook.

This Visual Basic for Applications example uses the **Logon** method to log on to a new <u>session</u>, displaying the dialog box to verify the profile name and password.

```
Set myOlApp = CreateObject("Outlook.Application")
Set myNameSpace = myOlApp.GetNameSpace("MAPI")
myNameSpace.Logon "myProfile", "myPassword", True, True
```

MarkComplete Method

Marks the task as completed. Sets PercentComplete to "100%", Complete to True , and DateCompleted to the current date.					
expression.MarkComplete					
<i>expression</i> Required. An expression that returns a TaskItem object.					

▼ Show All

Move Method

Moves an Outlook item to a new folder.

expression.Move(DestFldr)

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

DestFldr Required. An expression that returns a <u>MAPIFolder</u> object. The destination folder.

This Visual Basic for Applications example uses <u>GetDefaultFolder</u> to return the <u>MAPIFolder</u> object that represents the default folder. It then uses the <u>Find</u> and <u>FindNext</u> methods to find all messages sent by Mike Nash and uses the <u>Move</u> method to move all mail messages sent by Mike Nash from the default Inbox folder to the Personal Mail folder.

```
Set myOlApp = CreateObject("Outlook.Application")
Set myNameSpace = myOlApp.GetNameSpace("MAPI")
Set myInbox = myNameSpace.GetDefaultFolder(olFolderInbox)
Set myItems = myInbox.Items
Set myDestFolder = myInbox.Folders("Personal Mail")
Set myItem = myItems.Find("[SenderName] = 'Mike Nash'")
While TypeName(myItem) <> "Nothing"
    myItem.Move myDestFolder
    Set myItem = myItems.FindNext
Wend
```

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

```
Set myNameSpace = Application.GetNameSpace("MAPI")
Set myInbox = myNameSpace.GetDefaultFolder(6)
Set myItems = myInbox.Items
Set myDestFolder = myInbox.Folders("Personal Mail")
Set myItem = myItems.Find("[SenderName] = 'Mike Nash'")
While TypeName(myItem) <> "Nothing"
    myItem.Move myDestFolder
    Set myItem = myItems.FindNext
Wend
```

MoveTo Method

Moves a folder to the specified destination folder.

expression.MoveTo(DestinationFolder)

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

DestinationFolder Required. An expression that returns a **MAPIFolder** object. The destination folder for the folder that is being moved.

This Visual Basic for Applications example adds a new folder to the default Contacts folder, and then uses the **MoveTo** method to move it to the default Inbox folder.

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

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

```
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. On Status Change

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

This Microsoft Visual Basic/Visual Basic for Applications example shows how to call the **OnStatusChange** method to notify Microsoft Outlook that the user has changed a value on a custom property page.

```
Private Sub Option1_Click()
    Dim myPPSite As Outlook.PropertyPageSite
    Set myPPSite = Parent
    If Not TypeName(myPPSite) = "Nothing" Then
        globNewUserType = globAdministrator
        If globUserType <> globNewUserType Then
        globDirty = True
        myPPSite.OnStatusChange
        End If
    Else
        If TypeName(myPPSite) = "Nothing" Then
            MsgBox "The Property Page returned an empty result."
        End If
    End If
End Sub
```

PickFolder Method

The **PickFolder** method displays the Pick Folder dialog box. This is a modal dialog box which means that code execution will not continue until the user either selects a folder or cancels the dialog box. Returns a **MAPIFolder** object corresponding to the folder that the user selects in the dialog box. Returns **Nothing** when the dialog box is canceled by the user.

expression. PickFolder		
expression	Required. A NameSpace object.	

Post Method

Sends (posts) the PostItem object. The Post method, which is analogous to the Send method for the MailItem object, is used to save the post to the target public folder instead of mailing it.		
expression.Post		
expression	Required. An expression that returns a PostItem object.	

▼ Show All

PrintOut Method

Prints the Outlook <u>item</u> using all default settings.
Note The PrintOut method is the only Outlook method that can be used for printing.
expression.PrintOut
<i>expression</i> Required. An expression that returns one of the objects in the Applies To list.

▼ Show All

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 thus are accessible to everyone. The Personal form registry holds forms that are accessible only to the current store user.

Note The **Name** property must be set before you can use the **PublishForm** method.

expression.PublishForm(Registry, Folder)

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

Registry Required OlFormRegistry. The form class.

OlFormRegistry can be one of these OlFormRegistry constants.

olDefaultRegistry Handles items without regard to their form class.

olFolderRegistry

olOrganizationRegistry

olPersonalRegistry

Folder Optional except with **olFolderRegistry**. Expression that returns a **MAPIFolder** object. Used only with Folder form registry. The folder object from which the forms must be accessed.

This Visual Basic for Applications example creates a contact, obtains its **FormDescription** object, and saves it in the Folder form registry of the default Contacts folder.

Note The **PublishForm** method will return an error if the caption (**Name**) for the form is not set first.

```
Set myOlApp = CreateObject("Outlook.Application")
Set myNameSpace = myOlApp.GetNameSpace("MAPI")
Set myFolder = _
         myNameSpace.GetDefaultFolder(olFolderContacts)
Set myItem = myOlApp.CreateItem(olContactItem)
Set myForm = myItem.FormDescription
myForm.Name = "My Contact"
myForm.PublishForm olFolderRegistry, myFolder
```

This Visual Basic for Applications example creates an appointment, obtains its **FormDescription** object, and saves it in the user's Personal form registry.

To view the form after you have published it, go to **Choose Form** in the **File**, **New** menu and choose **Personal Forms Library** in the **Look In**: box. You can double-click on your new form, "Interview Scheduler" in the list.

```
Set myOlApp = CreateObject("Outlook.Application")
Set myItem = myOlApp.CreateItem(olAppointmentItem)
Set myForm = myItem.FormDescription
myForm.Name = "Interview Scheduler"
myForm.PublishForm olPersonalRegistry
```

▼ Show All

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 Application objection	ect.	

▼ Show All

Remove Method

_

▶ Remove method as it applies to the <u>Actions</u>, <u>Attachments</u>, <u>Folders</u>, <u>Items</u>, <u>ItemProperties</u>, <u>Pages</u>, <u>Recipients</u>, and <u>UserProperties</u> 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.

▶ <u>As it applies to the **Actions**, **Attachments**, **Folders**, **Items**, **ItemProperties**, **Pages**, **Recipients**, and **UserProperties** objects.</u>

This Visual Basic for Applications example uses the **Remove** method to remove all attachments from a forwarded message before sending it on to Jeff Smith.

```
Set myOlApp = CreateObject("Outlook.Application")
Set myItem = myOlApp.ActiveInspector.CurrentItem.Forward
Set myAttachments = myItem.Attachments
While myAttachments.Count > 0
          myAttachments.Remove 1
Wend
myItem.Recipients.Add "Jeff Smith"
myItem.Send
```

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

```
Set myItem = Application.ActiveInspector.CurrentItem.Forward
Set myAttachments = myItem.Attachments
While myAttachments.Count > 0
        myAttachments.Remove 1
Wend
myItem.Recipients.Add "Jeff Smith"
myItem.Send
```

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

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

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

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

RemoveMember Method

Removes an individual member from a given distribution list.

expression.RemoveMember(Recipient)

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

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

The following example removes a member from the distribution list called Group List. The **RemoveMember** method will fail if the specified recipient is not valid.

```
Sub RemoveRec()
'Remove a recipient from the list and displays new list.
    Dim olApp As Outlook.Application
    Dim objDstList As DistListItem
    Dim objName As NameSpace
    Dim objRcpnt As Recipient
    Dim objMail As MailItem
    Set olApp = Outlook.Application
    Set objName = olApp.GetNamespace("MAPI")
    Set objDstList = objName.GetDefaultFolder(olFolderContacts).Item
    Set objMail = olApp.CreateItem(olMailItem)
    Set objRcpnt = objMail.Recipients.Add(Name:="Jeff Smith")
    objDstList.RemoveMember Recipient:=objRcpnt
    objDstList.Display
    objDstList.Body = "Last Modified: " & Now
End Sub
```

RemoveMembers Method

Removes members from a distribution list.

expression.RemoveMembers(Recipients)

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

Recipients Required Recipients. The members to be removed from the distribution list.

This Microsoft Visual Basic/Visual Basic for Applications example locates every distribution list item in the default Contacts folder and then determines if the current user is a member of the list. If that is the case, the current user is removed from the distribution list.

```
Dim myOlApp As New Outlook.Application
Dim myNameSpace As Outlook.NameSpace
Dim myFolder As Outlook.MAPIFolder
Dim myDistList As Outlook.DistListItem
Dim tmpRecips As Outlook.Recipients
Set myNameSpace = myOlApp.GetNamespace("MAPI")
Set tmpRecips = myOlApp.CreateItem(olMailItem).Recipients
tmpRecips.Add myNameSpace.CurrentUser.Name
Set myFolder = myNameSpace.GetDefaultFolder(olFolderContacts)
For x = 1 To myFolder.Items.Count
    If TypeName(myFolder.Items.Item(x)) =  
            "DistListItem" Then
        Set myDistList = myFolder.Items.Item(x)
        For y = 1 To myDistList.MemberCount
            If myDistList.GetMember(y).Name =
                    myNameSpace.CurrentUser.Name Then
                myDistList.RemoveMembers tmpRecips
                myDistList.Save
                Exit For
            End If
        Next y
    End If
Next x
```

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

RemoveStore Method

Removes a folder from the Folder List in the user interface.

expression.RemoveStore(Folder)

expression Required. An expression that returns a Namespace object.

Folder Required **MAPIFolder** object. The folder to be deleted from the list.

Remarks

You cannot remove a store from the main mailbox on the server.

The following subroutine removes a folder called Personal Folders from the list of folders.

```
Sub RemovePST()

Dim objName As NameSpace
Dim objFolder As MAPIFolder

Set objName = Application.GetNamespace("MAPI")
Set objFolder = objName.Folders.Item("Personal Folders")
objName.RemoveStore objFolder

End Sub
```

Reply Method

Creates a reply, pre-addressed to the original sender, from the original message. Returns the reply as a MailItem object.		
expression.Reply		
<i>expression</i> 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
<i>expression</i> 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 example resets all built-in views in the user's Inbox to their original settings. The <u>Standard</u> property is returned to determine if the view is a built-in Outlook view.

```
Sub ResetViews()
'Resets all standard views in the user's Inbox

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

Set olApp = Outlook.Application
Set objName = olApp.GetNamespace("MAPI")
Set objViews = objName.GetDefaultFolder(olFolderInbox).Views
For Each objView In objViews
If objView.Standard = True Then
objView.Reset
End If
Next objView

End Sub
```

ResetColumns Method

The ResetColumns method clears the properties that have been cached with the SetColumns method. All properties are accessible after calling the ResetColumns method. SetColumns should be reused to store new properties again. ResetColumns does nothing if SetColumns has not been called first.		
expression.ResetColumns		
<i>expression</i> Required. An expression that returns one of the objects in the Applies To list.		

▼ Show All

Resolve Method

-

Attempts to resolve a **Recipient** object against the Address Book. Returns **True** if the object was resolved, **False** if it was not.

expression. Resolve

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

This Visual Basic for Applications example uses the **Resolve** method to attempt to resolve a recipient and, if unsuccessful, displays the <u>item</u> so a user can resolve it manually.

```
Set myOlApp = CreateObject("Outlook.Application")
Set myItem = myOlApp.CreateItem(olMailItem)
Set myRecipient = myItem.Recipients.Add("Tim O'Brien")
If Not myRecipient.Resolve Then myItem.Display
```

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

```
Set myItem = Application.CreateItem(0)
Set myRecipient = myItem.Recipients.Add("Tim O'Brien")
If Not myRecipient.Resolve Then myItem.Display
```

ResolveAll Method

Attempts to resolve all the **Recipient** objects in the **Recipients** collection against the Address Book. Returns **True** if all of the objects were resolved, **False** if one or more were not.

expression. ResolveAll

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

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

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.

Respond Method

_

Responds to a meeting request for the **AppointmentItem** object or a task request for the **TaskItem** object.

expression.Respond(Response, fNoUI, fAdditionalTextDialog)

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

Response Required OlMeetingResponse. The response to the request.

OlMeetingResponse can be one of these OlMeetingResponse constants.

For an **AppointmentItem** object:

olMeetingAccepted

olMeetingDeclined

olMeetingTentative

For a **TaskItem** object:

olTaskAccept

olTaskAssign

olTaskDecline

olTaskSimple

fNoUI Optional for **AppointmentItem**, required for **TaskItem**. **Boolean**. **True** to not display a dialog box; the response is sent automatically. **False** to display the dialog box for responding.

fAdditionalTextDialog Optional for **AppointmentItem**, required for **TaskItem**. **Boolean**. **False** to not prompt the user for input; the response is displayed in the inspector for editing. **True** to prompt the user to either send or send with comments. This argument is valid only if **fNoUI** is **False**.

Note The possible values for the optional parameters, *fNoUI* and

fAdditionalTextDialog and the subsequent results are as follows:

fNoUI, fAdditionalTextDialog	Result For AppointmentItem and TaskItem :		
True, True	Response item is returned with no user interface. To send the response, you must call the <u>Send</u> method.		
Those False	For AppointmentItem and TaskItem :		
True, False	Same result as with True , True .		
	For AppointmentItem :		
	Prompts user to Send or Edit before sending response.		
False, True	For TaskItem :		
	If Display method has been called, the user prompt appears. Otherwise, the item is sent without prompting and resulting item is nothing.		
	For AppointmentItem :		
False, False	New response item appears in the user interface, but no prompt.		
1 alse, 1 alse	For TaskItem :		
	Does nothing.		

This Visual Basic for Applications example uses the **Find** method to locate a **MeetingItem** in the user's Inbox. If there are no meeting items, the user is informed. If a meeting item is located, the **GetAssociatedAppointment** method is called to get the **AppointmentItem** and the **Respond** method allows acceptance.

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

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

Restrict Method

-

Applies a filter to the <u>Items</u> collection, returning a new collection containing all items from the original which match the filter. This method is an alternative to using the <u>Find</u> method or <u>FindNext</u> method to iterate over specific items within a collection. The **Find** or **FindNext** methods are faster than filtering if there are a small number of items. The **Restrict** method is significantly faster if there is a large number of items in the collection, especially if only a few items in the large collection are expected to be found.

Note If you are using user-defined fields as part of a **Find** or **Restrict** clause, the user-defined fields must exist in the folder, otherwise the code will generate an error stating that the field is unknown. You can add a field to a folder by displaying the **Field Chooser** and clicking **New**.

expression.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

LastFirstNoSpaceCompany

Categories

LastFirstSpaceOnly

Children

LastFirstSpaceOnlyCompany

Class

MemberCount

Companies

NetMeetingAlias

Company Last First No Space

NetMeetingAutoStart

 ${\bf Company Last First Space Only}$

NetMeetingOrganizerAlias

ContactNames

NetMeetingServer

Contacts

 ${\bf Net Meeting Type}$

ConversationIndex

RecurrenceState

DLName

ReplyRecipients

Email1EntryID

 ${\bf Received By Entry ID}$

Email2EntryID

Receved On Behalf Of Entry ID

Email3EntryID

Response State

EntryID

Saved

HTMLBody

Sent

IsOnlineMeeting

Submitted

LastFirstAndSuffix

VotingOptions

LastFirstNoSpace

This Visual Basic for Applications example uses the **Restrict** method to get all Inbox items dealing with Project X and moves them to the Project X folder.

```
Set myOlApp = CreateObject("Outlook.Application")
Set myNameSpace = myOlApp.GetNameSpace("MAPI")
Set myFolder = _
         myNameSpace.GetDefaultFolder(olFolderInbox)
Set myItems = myFolder.Items
Set myRestrictItems = myItems.Restrict _ ("[Categories] = 'Proje
For Each myItem In myRestrictItems
         myItem.Move myFolder.Folders("Project X")
Next
```

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

```
Set myNameSpace = Application.GetNameSpace("MAPI")
Set myFolder = myNameSpace.GetDefaultFolder(6)
Set myItems = myFolder.Items
Set myRestrictItems = myItems.Restrict _ ("[Categories] = 'Proje
For Each myItem In myRestrictItems
    myItem.Move myFolder.Folders("Project X")
Next
```

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

The following Visual Basic for Applications example is the same as the example above, except that it demonstrates the use of a variable in the filter.

```
Public Sub ContactDateCheck()
   Set myOlApp = CreateObject("Outlook.Application")
   Set myNameSpace = myOlApp.GetNamespace("MAPI")
   Set myContacts = myNameSpace.GetDefaultFolder(olFolderContacts).I
   DateStart = #6/11/97#
   DateToCheck$ = "[LastModificationTime] >= """ & DateStart & """"
   Set myRestrictItems = myContacts.Restrict(DateToCheck$)
   For Each myItem In myRestrictItems
        MsgBox myItem.FullName & ": " & MyItem.LastModificationTime
   Next
End Sub
```

▼ Show All

Save Method

_

Saves the Outlook <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.

This Visual Basic for Applications example creates a task called "Oil Change" that recurs every three months and uses the **Regenerate** property to set it to regenerate after each recurrence.

```
Set myOlApp = CreateObject("Outlook.Application")
Set myItem = myOlApp.CreateItem(olTaskItem)
Set myPattern = myItem.GetRecurrencePattern
myPattern.RecurrenceType = olRecursMonthly
myPattern.Regenerate = True
myPattern.Interval = 3
myItem.Subject = "Oil Change"
myItem.Save
myItem.Display
```

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

```
Set myItem = Application.CreateItem(3)
Set myPattern = myItem.GetRecurrencePattern
myPattern.RecurrenceType = 2
myPattern.Regenerate = True
myPattern.Interval = 3
myItem.Subject = "Oil Change"
myItem.Save
myItem.Display
```

▼ Show All

SaveAs Method

Saves the Outlook <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 is used.

expression.SaveAs(Path, Type)

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

Path Required **String**. The path in which to save the item.

Type Optional **Variant**. The file type to save. Can be one of the following **OlSaveAsType** constants: **olDoc**, **olHTML**, **olMSG**, **olRTF**, **olTemplate**, **olTXT**, **olVCal**, or **olVCard**.

This Visual Basic for Applications example uses the <u>SaveAs</u> method to save the currently open <u>item</u> as a text file in the My Documents folder, using the subject as the file name.

```
Dim myItem As Inspector
Dim objItem As Object
Set myOlApp = CreateObject("Outlook.Application")
Set myItem = myOlApp.ActiveInspector
If Not TypeName(myItem) = "Nothing" Then
        Set objItem = myItem.CurrentItem
        strname = objItem.Subject
        objItem.SaveAs "C:\" & " & strname & .txt", olTXT
Else
        MsgBox "There is no current active Inspector."
End If
```

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

```
Set myItem = Application.ActiveInspector
  If Not TypeName(myItem) = "Nothing" Then
        Set objItem = myItem.CurrentItem
        strname = objItem.Subject
        objItem.SaveAs "C:\" & " & strname & .txt", 0
  Else
        MsgBox "There is no current active Inspector."
  End If
```

▼ Show All

SaveAsFile Method

Saves the attachment to the specified path.

expression.SaveAsFile(Path)

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

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

This Visual Basic for Applications example uses the **SaveAsFile** method to save the first attachment of the currently open <u>item</u> as a file in the My Documents folder, using the attachment's display name as the file name.

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

Send Method

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

expression.Send

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

Example

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

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

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

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

SetColumns Method

The **SetColumns** method allows the user to cache certain properties for extremely fast access to those particular properties of the item. The **SetColumns** method is useful for iterating through the **Items** object. If you don't use this method, Microsoft Outlook must open each item to access the property. With the **SetColumns** method, Outlook only checks the properties that you have cached. Properties which are not cached are returned empty.

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

Sent Sent

DLName

Submitted				
VotingOptions				

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 VBScript example uses the **Open** event to display the "All Fields" page every time the item is opened.

	•	() cor.SetCur	rentForm	nPage "A]	l Fields	S"	

SetIcon Method

Sets the icon for the specified Microsoft Outlook bar shortcut.

 $expression. \textbf{SetIcon}(\textbf{\textit{Icon}})$

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

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

The following example sets the icon of all shortcuts with the target **MAPIFolder** object to the icon image Mail.ico located on the user's computer. The example assumes that this icon exists in the specified location.

```
Sub SetFolderIcon()
    Dim OlApp As New Outlook.Application
    Dim obj0lBar As Outlook.OutlookBarPane
    Dim objolGroup As Outlook.OutlookBarGroup
    Dim obj0lShortcuts As Outlook.OutlookBarShortcuts
    Dim obj0lShortcut As Outlook.OutlookBarShortcut
    Dim intTop As Integer
    Dim i As Integer
    Set obj0lBar = OlApp.ActiveExplorer.Panes.Item("OutlookBar")
    Set objolGroup = obj0lBar.Contents.Groups.Item(1)
    Set obj0lShortcuts = obj0lGroup.Shortcuts
    intTop = obj0lShortcuts.Count
    For i = intTop To 1 Step -1
        Set obj0lShortcut = obj0lShortcuts.Item(i)
        If TypeName(obj0lShortcut.Target) = "MAPIFolder" Then
            obj0lShortcut.SetIcon
                ("C:\Program Files\Microsoft Office\Office\forms\103
        End If
    Next i
End Sub
```

ShowCategoriesDialog Method

Displays the **Show Categories** dialog box, which allows you to select categories that correspond to the subject of the item.

expression. Show Categories Dialog

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

The following example creates a new appointment item, displays the item on the screen, and opens up the **Show Categories** dialog box.

```
Sub Appointment()
'Creates an appointment item to access Fields property
    Dim appolApp As Outlook.Application
    Dim olApptItem As AppointmentItem
    'Create an instance of the application
    Set appolApp = Outlook.Application
    'Create appointment item
    Set olApptItem = appolApp.CreateItem(olAppointmentItem)
    olApptItem.Body = "Please meet with me regarding these sales fig
    olApptItem.Recipients.Add ("Jeff Smith")
    olApptItem.Subject = "Low Sales Reports"
    'Display the item
    olApptItem.Display
    'Display the Show categories dialog
    olApptItem.ShowCategoriesDialog
End Sub
```

▼ Show All

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 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.

```
Set myOlApp = CreateObject("Outlook.Application")
Set myInspector = myOlApp.ActiveInspector
On Error GoTo ErrorHandler
myInspector.ShowFormPage("All Fields")
Set myItem = myInspector.CurrentItem
myItem.Display
Exit Sub
ErrorHandler:
    MsgBox "No current item to display!"
```

▼ Show All

ShowPane Method

Displays or hides a specific pane in the explorer.

Note You can also use the **Visible** property of the **OutlookBarPane** object to display or hide the **Outlook Bar**.

expression.ShowPane(Pane, Visible)

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

Pane Required OlPane.

OlPane can be one of these OlPane constants.

olFolderList olOutlookBar olPreview

Visible Required. **True** to make the pane visible, **False** to hide the pane.

This Microsoft Visual Basic/Visual Basic for Applications example uses the **ShowPane** and **IsPaneVisible** methods to hide the preview pane if it is visible, or to display it if it is hidden.

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

```
Set myOlExp = Application.ActiveExplorer
myOlExp.ShowPane 3, Not myOlExp.IsPaneVisible(3)
```

SkipRecurrence Method

Clears the current instance of a recurring task and sets the recurrence to the next instance of that task.	
expression.SkipRecurrence	
<i>expression</i> Required. An expression that returns a TaskItem object.	

Snooze Method

Delays the reminder by a specified time. This is equivalent to the user clicking the **Snooze** button.

expression.Snooze(SnoozeTime)

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

SnoozeTime Optional **Variant**. Indicates the amount of time (in minutes) to delay the reminder. The default value is 5 minutes.

Remarks

This method will fail if the current reminder is not active.

The following example delays all active reminders by a specified amount of time.

Sort Method

_

Sorts the collection of items by the specified property. The index for the collection is reset to 1 upon completion of this method.

expression.Sort(Property, Descending, Order)

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

Property Required **String**. The name of the property by which to sort, which may be enclosed in brackets (for example, "[CompanyName]"). May not be a user-defined field, and may not be a multi-valued property, such as a category.

Descending Applies to all objects in the Applies To list except the **AddressEntries** object. Optional **Variant** for the **Results** object; optional **Boolean** for all other objects. **True** to sort in descending order. The default value is **False** (ascending).

Order Applies to the **AddressEntries** object only. Optional **Variant**. The order for the specified address entries. Can be one of these **OlSortOrder** constants: **olAscending**, **olDescending**, or **olSortNone**.

Remarks

For the <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

 $Company Last First Space Only\ Net Meeting Organizer Alias$

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 example uses the **Sort** method to sort the **Items** collection for the default Contacts folder by the "CompanyName" property and then displays the company names each in turn.

```
Set myOlApp = CreateObject("Outlook.Application")
Set myNamespace = myOlApp.GetNamespace("MAPI")
Set myFolder = _
         myNameSpace.GetDefaultFolder(olFolderContacts)
Set myItems = myFolder.Items
myItems.Sort "[CompanyName]", False
For Each myItem in myItems
         MsgBox myItem.CompanyName
Next myItem
```

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

```
Set myNamespace = Application.GetNamespace("MAPI")
Set myFolder = _
          myNameSpace.GetDefaultFolder(10)
Set myItems = myFolder.Items
myItems.Sort "[CompanyName]", False
For Each myItem in myItems
          MsgBox myItem.CompanyName
Next
```

Start Method

Begins synchronizing a user's folders using the specified synchronization profile.

expression.Start

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

This Microsoft Visual Basic/Visual Basic for Applications example starts synchronization of the user's second synchronization profile in response to the user clicking a button on a form. The mySync0bj variable is declared as a public variable so it can be referenced by other modules.

```
Public mySyncObj As Outlook.SyncObject

Private Sub Command1_Click()
    Dim myOlApp As New Outlook.Application
    Dim mySyncs As Outlook.SyncObjects
    Set mySyncs = myOlApp.GetNamespace("MAPI").SyncObjects
    Set mySyncObj = mySyncs.Item(1)
    mySyncObj.Start
End Sub
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
<i>expression</i> Required. An expression that returns a JournalItem object.

StatusReport Method

_

Sends a status report to all CC recipients (recipients returned by the StatusUpdateRecipients property) with the current status for the task. Returns an **Object** representing the status report.

expression. Status Report

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

This Visual Basic for Applications example uses the **StatusReport** method to report the status of the currently open task.

```
Set myOlApp = CreateObject("Outlook.Application")
Set myinspector = myOlApp.ActiveInspector
If Not TypeName(myinspector) = "Nothing" Then
    If TypeName(myinspector.CurrentItem) = "TaskItem" Then
        Set myTask = myinspector.CurrentItem
        Set myReport = myTask.StatusReport
        myReport.Send
    End If
MsgBox "There is no task item currently open"
End If
MsgBox "There is no Inspector currently open"
```

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

```
Set myinspector = Application.ActiveInspector
If Not TypeName(myinspector) = "Nothing" Then
    If TypeName(myinspector.CurrentItem) = "TaskItem" Then
        Set myTask = myinspector.CurrentItem
        Set myReport = myTask.StatusReport
        myReport.Send
    End If
MsgBox "There is no task item currently open"
End If
MsgBox "There is no Inspector currently open"
```

Stop Method

Immediately ends synchronizing a user's folders using the specified synchronization profile. This method does not undo any synchronization that has already occurred.

expression.End

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

This Microsoft Visual Basic/Visual Basic for Applications example interrupts synchronization of the user's second synchronization profile in response to the user clicking a button on a form. The mySync0bj variable is declared as a public variable so it can be referenced by other modules.

```
Public mySyncObj As Outlook.SyncObject

Private Sub Command2_Click()
   MsgBox "Synchronization stopped by user."
   mySyncObj.Stop
End Sub
```

StopTimer Method

Stops the timer on the journal entry. This method allows programmatic control of the timer function. The <u>Duration</u> , <u>End</u> and <u>Start</u> properties are automatically updated when the timer is stopped.
expression.StopTimer
<i>expression</i> Required. An expression that returns a JournalItem object.

Update Method

The **Update** method posts a change to the **AddressEntry** object in the messaging system.

expression. Update(MakePermanent, Refresh)

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

MakePermanent Optional **Variant**. A value of **True** indicates that the property cache is flushed and all changes are committed in the underlying address book. A value of **False** indicates that the property cache is flushed but not committed to persistent storage. The default value is **True**.

Refresh Optional **Variant**. A value of **True** indicates that the property cache is reloaded from the values in the underlying address book. A value of **False** indicates that the property cache is not reloaded. The default value is **False**.

Remarks

New entries or changes to existing entries are not persisted in the collection until the **Update** method has been called with its *MakePermanent* parameter set to **True**.

To flush the cache and then reload the values from the address book, call **Update** with the *MakePermanent* parameter set to **False** and the *Refresh* parameter set to **True**.

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

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

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

Account Property

Returns or sets a String representing the account for the contact. Read/write.		
expression.	Account	
expression	Required. An expression that returns a ContactItem object.	

▼ Show All

Actions Property

-

Returns an **Actions** collection that represents all the available actions for the Outlook <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 example creates a new mail message and uses the **Add** method to add an **Action** to it.

```
Set myOlApp = CreateObject("Outlook.Application")
Set myItem = myOlApp.CreateItem(olMailItem)
Set myAction = myItem.Actions.Add
```

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

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

ActualWork Property

Returns or s Read/write.	ets a Long indicating the actual effort (in minutes) spent on the task.
expression.A	ActualWork
expression	Required. An expression that returns a TaskItem object.

Address Property

Returns or sets a String representing the e-mail address of the recipient. Read/write for the AddressEntry object; read-only for the Recipient object.
Note The Address property must be set before calling the Details method.
expression.Address
<i>expression</i> Required. An expression that returns one of the objects in the Applies To list.

AddressBookName Property

Returns or sets a **String** that indicates the Address Book name for the current **MAPIFolder** object. Read/write.

expression. AddressBookName

expression Required. An expression that returns a MAPIFolder object.

The following example changes the Address Book name for a given folder and displays the new name to the user. The subroutine accepts the folder object and a **String** representing the new folder name.

```
Sub BookName()
    Dim olApp As Outlook.Application
    Dim nmsName As NameSpace
    Dim fldFolder As MAPIFolder
    Dim strAns As String
    Set olApp = Outlook.Application
    'Create a reference to namepsace
    Set nmsName = olApp.GetNamespace("MAPI")
    'Create an instance of the Inbox folder
    Set fldFolder = nmsName.GetDefaultFolder(olFolderInbox)
    'Prompt user for input
    strAns = InputBox("Enter the name of the new address book")
    'Call Sub procedure
    Call Changebook(fldFolder, strAns)
End Sub
Sub Changebook(ByRef fldFolder As MAPIFolder, ByVal strName As Strin
'Changes the name of the address book for a given folder
    'Set address book name to user input
    fldFolder.AddressBookName = strName
    'Display message to user
    MsgBox ("The new address book name for the " & fldFolder.Name &
             & strName & ".")
End Sub
```

AddressEntries Property

Returns the AddressEntries collection for the specified object.	
expression.AddressEntries	
expression	Required. An expression that returns an AddressList object.

AddressEntry Property

Accessing the **AddressEntry** property forces resolution of an unresolved recipient name. If the name cannot be resolved, an error is returned. If the recipient is resolved, the **Resolved** property is **True**.

**expression.AddressEntry*

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

Returns the **AddressEntry** object corresponding to the resolved recipient.

▼ Show All

AddressLists Property

Returns an <u>AddressLists</u> 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 <u>AddressList</u> object represents one of the available address books. The type of access you obtain depends on the access permissions granted to you by each individual address book provider.

expression.AddressLists

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

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

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

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

AllDayEvent Property

True if the appointment is an all-day event (as opposed to a specified time). Read/write Boolean .	
expression.A	AllDayEvent
expression	Required. An expression that returns an AppointmentItem object.

AlternateRecipientAllowed Property

Anniversary Property

Returns or sets a Date indicating the anniversary date for the contact. Read/write.	
expression.Anniversary	
<i>expression</i> Required. An expression that returns a ContactItem object.	

AnswerWizard Property

Returns the AnswerWizard object for the application.		
expression.AnswerWizard()		
expression Re	equired. An expression that returns an Application object.	

AppFolders Property

This property returns the "Application Folders" **SyncObject**. The **SyncObject** is where folders are automatically added when the **InAppFolderSyncObject** property of the **MapiFolder** object is set to **True**. The **SyncObject** allows users to synchronize Microsoft Outlook folders, address books, and folder home pages for offline use.

expression.AppFolders

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

Example

The following example sets the **SyncObject** for the application folders and synchronizes the user's Inbox.

```
Public Sub SetAppfolders()

Dim olApp As New Outlook.Application
Dim nsp As Outlook.NameSpace
Dim objSycs As Outlook.SyncObjects
Dim objSyc As Outlook.SyncObject
Dim mpfInbox As Outlook.MAPIFolder

Set nsp = olApp.GetNamespace("MAPI")
Set objSycs = nsp.SyncObjects
Set objSyc = objSycs.AppFolders
Set mpfInbox = nsp.GetDefaultFolder(olFolderInbox)

mpfInbox.InAppFolderSyncObject = True
objSyc.Start

End Sub
```

Application Property

Returns an **Application** object that represents the parent application (Microsoft Outlook) for an object.

expression.Application

expression Required. An expression that returns an Outlook object.

Example

This Visual Basic for Applications example uses the **Application** property to access Outlook and then creates a new **MailItem** and displays it.

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

If you use VBScript, you do not use the **Application** property to retrieve the **Application** object. Instead, you reference the **Application** object directly. This example shows how to perform the same task using VBScript.

```
Set myItem = Application.CreateItem(0)
myItem.Display
```

AppointmentItem Property

Returns the **AppointmentItem** object that is the exception. Not valid for deleted appointments.

expression. AppointmentItem

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

Example

This Visual Basic for Applications example uses <u>CreateItem</u> to create an <u>AppointmentItem</u> object. The <u>RecurrencePattern</u> is obtained for this item using the <u>GetRecurrencePattern</u> method. By setting the <u>RecurrencePattern</u> properties, <u>RecurrenceType</u>, <u>PatternStartDate</u>, and <u>PatternEndDate</u>, the appointments are now a recurring series that occur on a daily basis for the period of one year.

An **Exception** object is created when one instance of this recurring appointment is obtained using the **GetOccurrence** method and properties for this instance are altered. This exception to the series of appointments is obtained using the **GetRecurrencePattern** method to access the **Exceptions** collection associated with this series. Message boxes display the original **Subject** and **OriginalDate** for this exception to the series of appointments and the current date, time, and subject for this exception.

For a description of changes required for this example to work in VBScript, see the Note at the end of the example.

```
Public Sub cmdExample()
    Set myOlApp = New Outlook.Application
    Set myApptItem = myOlApp.CreateItem(olAppointmentItem)
   myApptItem.Start = #2/2/1998 3:00:00 PM#
    myApptItem.End = #2/2/1998 4:00:00 PM#
   myApptItem.Subject = "Meet with Boss"
    'Get the recurrence pattern for this appointment
    'and set it so that this is a daily appointment
    'that begins on 2/2/98 and ends on 2/2/99
     'and save it.
    Set myRecurrPatt = myApptItem.GetRecurrencePattern
   myRecurrPatt.RecurrenceType = olRecursDaily
   myRecurrPatt.PatternStartDate = #2/2/1998#
    myRecurrPatt.PatternEndDate = #2/2/1999#
   myApptItem.Save
    'Access the items in the Calendar folder to locate
    'the master AppointmentItem for the new series.
    Set myNameSpace = myOlApp.GetNamespace("MAPI")
    Set myFolder = myNameSpace.GetDefaultFolder(olFolderCalendar)
```

```
Set myItems = myFolder.Items
    Set myApptItem = myItems("Meet with Boss")
    'Get the recurrence pattern for this appointment
    'and obtain the occurrence for 3/12/98.
    myDate = #3/12/1998 3:00:00 PM#
    Set myRecurrPatt = myApptItem.GetRecurrencePattern
    Set myOddApptItem = myRecurrPatt.GetOccurrence(myDate)
    'Save the existing subject. Change the subject and
    'starting time for this particular appointment
    'and save it.
    saveSubject = myOddApptItem.Subject
    myOddApptItem.Subject = "Meet NEW Boss"
    newDate = #3/12/1998 3:30:00 PM#
    myOddApptItem.Start = newDate
    myOddApptItem.Save
    'Get the recurrence pattern for the master
    'AppointmentItem. Access the collection of
    'exceptions to the regular appointments.
    Set myRecurrPatt = myApptItem.GetRecurrencePattern
    Set myException = myRecurrPatt.Exceptions.Item(1)
    'Display the original date, time, and subject
    'for this exception.
    MsgBox myException.OriginalDate & ": " & saveSubject
    'Display the current date, time, and subject
    'for this exception.
    MsgBox myException.AppointmentItem.Start & ": " & _
myException.AppointmentItem.Subject
End Sub
```

Note For this example to work properly in VBScript, there are only a few simple changes that need to be made in the code.

You don't have to retrieve the application as an object and you must use the values of the constants, so:

```
Set myOlApp = New Outlook.Application
    Set myApptItem = myOlApp.CreateItem(olAppointmentItem)
becomes:
Set myApptItem = Application.CreateItem(1)
```

```
and
myRecurrPatt.RecurrenceType = olRecursDaily
becomes:
myRecurrPatt.RecurrenceType = 0
and
Set myFolder = myNamespace.GetDefaultFolder(olFolderCalendar)
becomes:
Set myFolder = myNamespace.GetDefaultFolder(9)
```

Assistant Property

Some of the content in this topic may not be applicable to some languages.
Returns an Assistant object that represents the Microsoft Office Assistant.
expression.Assistant
<i>expression</i> Required. An expression that returns one of the objects in the Applies To list.

AssistantName Property

Returns or sets a String representing the name of the person who is the assistant for the contact. Read/write.
expression.AssistantName
expression Required. An expression that returns a ContactItem object.

AssistantTelephoneNumber Property

▼ Show All

Attachments Property

-

Returns an **Attachments** object that represents all the attachments for the specified <u>item</u>.

expression.Attachments

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

Example

This Visual Basic for Applications example uses the **Remove** method to remove all attachments from a forwarded message before sending it on to John Y. Chen.

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

AutoForwarded Property

True if the mail message was automatically forwarded. Read/write Boolean .
expression.AutoForwarded
<i>expression</i> Required. An expression that returns one of the objects in the Applies To list.

AutoResponse Property

Returns or sets a String representing the text of an automatic response for a Recipient . Read/write.		
expression.AutoResponse		
<i>expression</i> Required. An expression that returns a Recipient object.		

BCC Property

Returns a String representing the display list of blind carbon copy (BCC) names for a MailItem . This property contains the display names only. The Recipients collection should be used to modify the BCC recipients. Read/write.		
expression.BCC		
<i>expression</i> Required. An expression that returns a MailItem object.		

▼ Show All

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
<i>expression</i> Required. An expression that returns one of the objects in the Applies To list.

Birthday Property

Returns or sets a Date indicating the birthday for the contact. Read/write.		
expression.	Birthday	
expression	Required. An expression that returns a ContactItem object.	

▼ Show All

Body Property

Returns or sets a **String** representing the clear-text body of the Outlook <u>item</u>. Read/write.

Note The **EditorType** property is not affected when you merely access the **Body** property of the item (as in MsgBox myItem.Body), but when you reset the **Body** property (as in myItem.Body = "This is a new body"), the **EditorType** reverts back to the user's default editor.

expression.Body

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

Example

This VBScript example uses the **Open** event of an **item** to set its **Body** property.

```
Function Item_Open()
   Item.Body = "This is the message body."
End Function
```

▼ Show All

BodyFormat Property

Returns or sets an **OlBodyFormat** constant indicating the format of the body text. The body text format determines the standard used to display the text of the message. Microsoft Outlook provides three body text format options: Plain Text, Rich Text and HTML. Read/write **OlBodyFormat**.

OlBodyFormat can be one of these OlBodyFormat constants.

olFormatHTML olFormatPlain olFormatRichText olFormatUnspecified

expression.BodyFormat

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

Remarks

All text formatting will be lost when the **BodyFormat** property is switched from RTF to HTML and vice-versa.

The property can not be set to **olFormatUnspecified**, however it will return **olFormatUnspecified** if the item has not yet been displayed.

Example

The following example creates a new **MailItem** object and sets the **BodyFormat** property to **olFormatRichText**. The Body text of the Mail item will now appear in Rich Text format.

```
Sub NewMail()
'Creates a new Mailitem and modifies its properties

Dim olApp As Outlook.Application
Dim objMail As MailItem
Set olApp = Outlook.Application
'Create mail item
Set objMail = olApp.CreateItem(olMailItem)

With objMail
    .DownloadState = olHeaderOnly
    'Set body format to Rich Text
    .BodyFormat = olFormatRichText
    .Display
End With

End Sub
```

Business2TelephoneNumber Property

Returns or sets a String representing the second business telephone number the contact. Read/write.	for
expression.Business2TelephoneNumber	
<i>expression</i> Required. An expression that returns a ContactItem object.	

BusinessAddress Property

r		
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

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

Remarks

1 1 5 1	<u>BusinessAddress</u> property, but may be changed I it be parsed incorrectly. Note that any such
1 5	ty will be overwritten by any subsequent
changes or entries to the Busine	

BusinessAddressCountry Property

Returns or sets a **String** representing the country/region code portion of the business address for the contact. Read/write.

expression. Business Address Country

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

Remarks

This property is parsed from the BusinessAddress property, but may be change or entered independently should it be parsed incorrectly. Note that any such	ed
hanges or entries to this property will be overwritten by any subsequent	
hanges or entries to the BusinessAddress property.	

BusinessAddressPostalCode Property

Returns or sets a **String** representing the postal code (zip code) portion of the business address for the contact. Read/write.

expression. BusinessAddressPostalCode

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

Remarks

This property is parsed from the BusinessAddress property, but may be change or entered independently should it be parsed incorrectly. Note that any such	ed
hanges or entries to this property will be overwritten by any subsequent	
hanges or entries to the BusinessAddress property.	

BusinessAddressPostOfficeBox Property

Returns or sets a **String** representing the post office box number portion of the business address for the contact. Read/write.

expression. BusinessAddressPostOfficeBox

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

Remarks

1 1 5 1	<u>BusinessAddress</u> property, but may be changed I it be parsed incorrectly. Note that any such
1 5	ty will be overwritten by any subsequent
changes or entries to the Busine	

BusinessAddressState Property

Returns or sets a **String** representing the state code portion of the business address for the contact. Read/write.

expression. BusinessAddressState

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

Remarks

1 1 5 1	<u>BusinessAddress</u> property, but may be changed I it be parsed incorrectly. Note that any such
1 5	ty will be overwritten by any subsequent
changes or entries to the Busine	

BusinessAddressStreet Property

Returns or sets a **String** representing the street address portion of the business address for the contact. Read/write.

expression. BusinessAddressStreet

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

Remarks

1 1 5 1	<u>BusinessAddress</u> property, but may be changed I it be parsed incorrectly. Note that any such
1 5	ty will be overwritten by any subsequent
changes or entries to the Busine	

BusinessFaxNumber Property

Returns or sets a String representing the business fax number for the contact. Read/write.	
expression .Bu	sinessFaxNumber
expression R	equired. An expression that returns a ContactItem object.

BusinessHomePage Property

Returns or sets a String representing the URL of the business Web page for the contact. Read/write.	
expression.BusinessHomePage	
expression Required. An expression that returns a ContactItem object.	

BusinessTelephoneNumber Property

Returns or sets a String representing the first business telephone number for the contact. Read/write.	
expression.BusinessTelephoneNumber	
<i>expression</i> Required. An expression that returns a ContactItem object.	

BusyStatus Property

Returns or sets an OlBusyStatus constant indicating the busy status of the user for the appointment. Read/write.
OlBusyStatus can be one of these OlBusyStatus constants. olBusy olFree olOutOfOffice olTentative
expression.BusyStatus
<i>expression</i> Required. An expression that returns an AppointmentItem object.

CallbackTelephoneNumber Property

Returns or sets a String representing the callback telephone number for the contact. Read/write.	
expression.CallbackTelephoneNumber	
<i>expression</i> Required. An expression that returns a ContactItem object.	

Caption Property

Returns a String representing the window caption (title bar text) of an explorer or inspector window. Read-only.	
expression.Caption	
expression object.	Required. An expression that returns an Explorer or Inspector

CardData Property

Returns or sets a String representing the text of the card data for the task. Read/write.	
expression.CardData	
<i>expression</i> Required. An expression that returns a TaskItem object.	

CarTelephoneNumber Property

Returns or sets a String representing the car telephone number for the contact. Read/write.	
expression.CarTe	elephoneNumber
expression Requ	ired. An expression that returns a ContactItem object.

▼ Show All

Categories Property

Returns or sets a String representing the categories assigned to the Outlook <u>item</u> . Read/write.
expression.Categories
<i>expression</i> Required. An expression that returns one of the objects in the Applies To list.

Category Property

	ets a String representing the category assigned to the form Read/write.	
expression .(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		
<i>expression</i> 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		
<i>expression</i> Required. An expression that returns a ContactItem object.		

Class Property

Returns an **OlObjectClass** constant indicating the object's class. Read-only.

OlObjectClass can be one of these OlObjectClass constants.

olAction

olActions

olAddressEntries

olAddressEntry

olAddressList

olAddressLists

olApplication

olAppointment

olAttachment

olAttachments

olAutoFormatRule

olAutoFormatRules

olCalendarViewField

olCalendarViewFields

olCardViewField

olCardViewFields

olClassCalendarView

olClassCardView

olClassIconView

olClassTableView

olClassTimelineView

olContact

olDistributionList

olDocument

olException

```
olExceptions
```

olExplorer

olExplorers

olFolder

olFolders

olFormDescription

olIconViewField

olIconViewFields

olInspector

olInspectors

olItemProperties

olItemProperty

olItems

olJournal

olLink

olLinks

olMail

olMeetingCancellation

olMeetingRequest

ol Meeting Response Negative

ol Meeting Response Positive

olMeeting Response Tentative

olNamespace

olNote

olObjects

olOutlookBarGroup

olOutlookBarGroups

olOutlookBarPane

olOutlookBarShortcut

olOutlookBarShortcuts

olOutlookBarStorage

olPages

olPanes

olPost olProp olProp

 ${\bf ol Property Pages}$

ol Property Page Site

olRecipient

olRecipients

olRecurrencePattern

olRemote

olReport

olResults

olSearch

olSelection

 ${\bf olSyncObject}$

 ${\bf ol Sync Objects}$

olTableViewField

ol Table View Fields

olTask

ol Task Request

ol Task Request Accept

olTaskRequestDecline

olTaskRequestUpdate

olTimelineViewField

ol Timeline View Fields

olUserProperties

olUser Property

olViews

expression.Class

expression Required. An expression that returns an Outlook object.

		_

Color Property

Returns or sets a **Long** indicating the color of the note. Can be one of the following **OlNoteColor** constants: **olBlue**, **olGreen**, **olPink**, **olWhite**, or **olYellow**. Read/write.

objNoteItem.Color

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

Example

This example creates a new **NoteItem** and sets the color to blue.

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

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

```
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 **Application** object.

Example

This Microsoft Visual Basic example displays the number of COM add-ins currently connected.

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

```
Sub Commandbutton1_Click()
   MsgBox "There are " & _
        Application.COMAddIns.Count & " COM add-ins."

End Sub
```

CommandBars Property

Returns a CommandBars collection object that represents all the menus and toolbars in the Explorer or Inspector .			
expression.	expression.CommandBars		
expression object.	Required. An expression that returns an Explorer or Inspector		

Comment Property

	Returns or sets a String representing the text of a comment assigned to the form description. Read/write.		
expression.	expression.Comment		
expression	Required. An expression that returns a FormDescription object.		

▼ Show All

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

CompanyLastFirstNoSpace Property

Returns a **String** representing the company name for the contact followed by the concatenated last name, first name, and middle name with no space between the last and first names. This property is parsed from the **CompanyName**, **LastName**, **FirstName** and **MiddleName** properties. Read-only.

Note The **LastName**, **FirstName**, and **MiddleName** properties are themselves parsed from the **FullName** property.

expression.CompanyLastFirstNoSpace

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

Note The value of this property is only filled when its associated property (**FirstName**, **LastName**, **MiddleName**, **CompanyName**, and **Suffix**) contain Asian (DBCS) characters. If the corresponding field does not contain Asian characters, the property will be empty.

CompanyLastFirstSpaceOnly Property

_

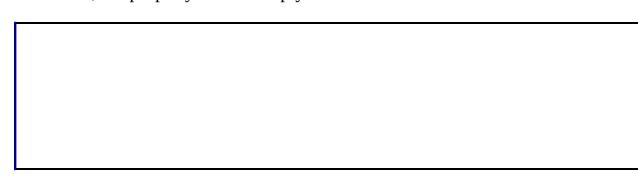
Returns a **String** representing the company name for the contact followed by the concatenated last name, first name, and middle name with spaces between the last, first, and middle names. This property is parsed from the **CompanyName**, **LastName**, **FirstName** and **MiddleName** properties. Read-only.

Note The **LastName**, **FirstName**, and **MiddleName** properties are themselves parsed from the **FullName** property.

expression.CompanyLastFirstSpaceOnly

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

Note The value of this property is only filled when its associated property (**FirstName**, **LastName**, **MiddleName**, **CompanyName**, and **Suffix**) contain Asian (DBCS) characters. If the corresponding field does not contain Asian characters, the property will be empty.



CompanyMainTelephoneNumber Property

-		
Returns or sets a String representing the company main telephone number for the contact. Read/write.		
expression.CompanyMainTelephoneNumber		
<i>expression</i> Required. An expression that returns a ContactItem object.		

CompanyName Property

Returns or sets a String representing the company name for the contact. Read/write.		
expression.CompanyName		
<i>expression</i> Required. An expression that returns a ContactItem object.		

Complete Property

True if the task is completed. Read/write Boolean .					
expression.	expression.Complete				
expression	Required. An expression that returns a TaskItem object.				

ComputerNetworkName Property

Returns or sets a String representing the name of the computer network for the contact. Read/write.				
expression.ComputerNetworkName				
<i>expression</i> Required. An expression that returns a ContactItem object.				
	_			

ConferenceServerAllowExternal Property

Reserv	ved for fut	ure use.			

ConferenceServerPassword Property

Reserved	for future u	ıse.			

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				
<i>expression</i> Required. An expression that returns a FormDescription object.				

ContactNames Property

Returns a String representing the contact names associated with the journal entry. This property contains the display names for the contacts only. Use the Recipients object to modify the contents of this string. Read/write.				
expression.ContactNames				
<i>expression</i> Required. An expression that returns a <u>JournalItem</u> or <u>TaskItem</u> object.				

Contents Property

-

Returns the **OutlookBarStorage** object for the specified **Outlook Bar** pane.

expression.Contents

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

Example

This Microsoft Visual Basic/Visual Basic for Applications example displays a message listing the groups in the Outlook Bar.

```
Dim myOlApp As New Outlook.Application
Dim myOlBar As Outlook.OutlookBarPane
Dim myOlGroups As Outlook.OutlookBarGroups
myMsg = "The groups in the Outlook Bar are:"
Set myOlBar = myOlApp.ActiveExplorer.Panes.Item("OutlookBar")
Set myOlGroups = myOlBar.Contents.Groups
For x = 1 To myOlGroups.Count
    myMsg = myMsg \& Chr(13) \& myOlGroups.Item(x)
Next x
MsgBox myMsg
If you use VBScript, you do not create the Application object. This example
shows how to perform the same task using VBScript.
myMsg = "The groups in the Outlook Bar are:"
Set myOlBar = Application.ActiveExplorer.Panes.Item("OutlookBar")
Set myOlGroups = myOlBar.Contents.Groups
For x = 1 To myOlGroups.Count
    myMsg = myMsg \& Chr(13) \& myOlGroups.Item(x)
Next
MsgBox myMsg
```

ConversationIndex Property

Returns a String representing the index of the conversation thread of the <u>item</u> . Read-only.					
expression.ConversationIndex					
<i>expression</i> 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 item created by the action. Read/write.

OlActionCopyLike can be one of these OlActionCopyLike constants.

olForward

olReply

olReplyAll

olReplyFolder

olRespond

expression.Copy

expression Required. An expression that returns an Action object.

Count Property

Returns a Long indicating the count of objects in the specified collection.					
expression.Count					
<i>expression</i> Required. An expression that returns a Microsoft Outlook collectiobject.	on				

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				
<i>expression</i> Required. An expression that returns one of the objects in the Applies To list.				

CurrentFolder Property

Returns or sets a **MAPIFolder** object that represents the current folder displayed in the explorer. Use this property to change the folder the user is viewing.

expression. CurrentFolder

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

Example

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

```
Set myOlApp = CreateObject("Outlook.Application")
Set myNameSpace = myOlApp.GetNameSpace("MAPI")
Set myOlApp.ActiveExplorer.CurrentFolder = _
    myNameSpace.GetDefaultFolder(olFolderCalendar)
```

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

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

CurrentGroup Property

Returns or sets an **OutlookBarGroup** object that represents the group that is open in the specified **Outlook Bar** pane.

expression. CurrentGroup

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

Remarks

You must use the Set keyword to set this property, as shown in this example.					
Set myOlPane.CurrentGroup = myOlGroup					
Setting this property causes the BeforeGroupSwitch event to occur.					

CurrentItem Property

Returns an **Object** representing the current <u>item</u> being displayed in the inspector.

Note If there is no currently open item, an error will be returned.

expression.CurrentItem

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

Example

This Visual Basic for Applications example uses the <u>CurrentItem</u> property to obtain the current <u>item</u> that the user is viewing. If there is no currently open item, an error will be returned.

Set myOlApp = CreateObject("Outlook.Application") On Error Resume Next Set myItem = myOlApp.ActiveInspector. CurrentItem							

CurrentUser Property

Returns the currently logged-on user as a **Recipient** object.

expression. CurrentUser

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

Example

This Visual Basic for Applications example uses the <u>CurrentUser</u> property to obtain the name of the currently logged-on user and then displays a message box containing the name.

```
Set myOlApp = CreateObject("Outlook.Application")
Set myNameSpace = myOlApp.GetNameSpace("MAPI")
MsgBox myNameSpace.CurrentUser.Name
```

Set myNameSpace = Application.GetNameSpace("MAPI")

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

```
MsgBox myNameSpace.CurrentUser.Name
```

CurrentView Property

Returns or sets an **Object** (for the **MAPIFolder** object) or **Variant** (for the **Explorer** object) representing the current <u>view</u>. Read/write.

expression. CurrentView

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

Remarks

When this property is set, two events occur: <u>BeforeViewSwitch</u> occurs before the actual view change takes place and can be used to cancel the change; <u>ViewSwitch</u> takes place after the change is effective.

Example

The following example sets the current view in the active explorer to Messages.

```
Dim myOlApp As New Outlook.Application
Dim myOlExp As Outlook.Explorer
Set myOlExp = myOlApp.ActiveExplorer
myOlExp.CurrentView = "Messages"
```

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

рттсат	LIUII.ACLI	veExhiole	er.Currei	irview –	"Messages	

CustomerID Property

CustomViewsOnly Property

Returns or sets a **Boolean** that determines which views are displayed on the **Views** menu for a given folder. If set to the **True**, only user created-views will appear on the menu. Read/write.

expression. CustomViewsOnly

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

Example

The following example prompts the user to select a view option. If the user chooses to view all views, the **CustomViewsOnly** property is set to **False**. If the user chooses to view only custom views, the **CustomViewsOnly** property is set to **True**. Once the property is changed, the outcome of the change is displayed for the user.

```
Sub SetCusView()
'Sets the CustomViewsOnly property depending on the user's response
    Dim olApp As Outlook.Application
    Dim nmsName As NameSpace
    Dim fldFolder As MAPIFolder
    Dim lngAns As Long
    Set olApp = Outlook.Application
    Set nmsName = olApp.GetNamespace("MAPI")
    Set fldFolder = nmsName.GetDefaultFolder(olFolderInbox)
    'Prompt user for input
    lngAns = MsgBox("Would you like to view only custom views in the
    Call SetVal(fldFolder, lngAns)
End Sub
Sub SetVal(ByRef fldFolder As MAPIFolder, ByVal lngAns As Long)
'Modifies the CustomViewsOnly property to display views on the Views
    If lngAns = vbYes Then
        fldFolder.CustomViewsOnly = True
    Else
        fldFolder.CustomViewsOnly = False
    End If
    'Display only custom views
    If lngAns = vbYes Then
       MsgBox "The " & fldFolder.Name & " folder will now display on
    'display all views
       MsgBox "The " & fldFolder.Name & " folder will now display al
    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

the day of the month on which the recurring write.
sion that returns a RecurrencePattern object.

DayOfWeekMask Property

Returns or sets an <u>OlDaysOfWeek</u> 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

ol We dnesday

expression. DayOfWeekMask

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

Example

This Visual Basic for Applications example uses <u>GetRecurrencePattern</u> to obtain the <u>RecurrencePattern</u> object for the newly-created <u>AppointmentItem</u>. The properties, <u>RecurrenceType</u>, <u>DayOfWeekMask</u>, <u>PatternStartDate</u>, <u>Interval</u>, <u>PatternEndDate</u>, and <u>Subject</u> are set, the appointment is saved and then displayed with the pattern: "Occurs every 3 week(s) on Monday effective 1/21/98 until 12/21/2001 from 2:00 PM to 5:00 PM."

```
Set myOlApp = New Outlook.Application
Set myApptItem = myOlApp.CreateItem(olAppointmentItem)
Set myRecurrPatt = myApptItem.GetRecurrencePattern

myRecurrPatt.RecurrenceType = olRecursWeekly
myRecurrPatt.DayOfWeekMask = olMonday
myRecurrPatt.PatternStartDate = #1/21/98 2:00:00 PM#
myRecurrPatt.Interval = 3
myRecurrPatt.PatternEndDate = #12/21/2001 5:00:00 PM#
myApptItem.Subject = "Important Appointment"
myApptItem.Save
myApptItem.Display
```

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

```
Set myApptItem = Application.CreateItem(1)
Set myRecurrPatt = myApptItem.GetRecurrencePattern
myRecurrPatt.RecurrenceType = 1
myRecurrPatt.DayOfWeekMask = 2
myRecurrPatt.PatternStartDate = #1/21/98 2:00:00 PM#
myRecurrPatt.Interval = 3
myRecurrPatt.PatternEndDate = #12/21/2001 5:00:00 PM#
myApptItem.Subject = "Important Appointment"
myApptItem.Save
myApptItem.Display
```

		_

▼ Show All

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

DefaultMessageClass Property

Returns a St Read-only.	ring representing the default message class for items in the folder.
expression . I	DefaultMessageClass
expression	Required. An expression that returns a MAPIFolder object.

DeferredDeliveryTime Property

Returns or sets a **Date** indicating the date and time the mail message is to be delivered. This property corresponds to the MAPI property PR_DEFERRED_DELIVERY_TIME. Read/write.

expression.DeferredDeliveryTime

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

DelegationState Property

Returns an OlTaskDelegationState constant indicating the delegation state of the task. Read-only.

OlTaskDelegationState can be one of these OlTaskDelegationState constants.

olTaskDelegationAccepted
olTaskDelegationDeclined
olTaskDelegationUnknown
olTaskNotDelegated

expression.DelegationState

expression Required. An expression that returns a TaskItem object.

Delegator Property

Returns a String representing the display name of the delegator for the task. Read-only.	
expression.Delegator	
<i>expression</i> Required. An expression that returns a TaskItem object.	
	_

DeleteAfterSubmit Property

True if a copy of the mail message is not saved upon being sent. False if a copy is saved. Read/write Boolean .
expression.DeleteAfterSubmit
<i>expression</i> 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
<i>expression</i> Required. An expression that returns an Exception object.

Department Property

Returns or so Read/write.	ets a String representing the department name for the contact.
expression . L	Department
expression	Required. An expression that returns a ContactItem object.

Description Property

	s a String representing the description of the folder. This property the MAPI property PR_COMMENT. Read/write.
expression .De	scription
expression F	Required. An expression that returns a MAPIFolder object.

Dirty Property

Returns **True** if the contents of a custom property page have been altered. The ActiveX control that implements the **PropertyPage** object sets the value of this property, and Microsoft Outlook queries this in response to the **OnStatusChange** method of a **PropertyPageSite** object. Read-only **Boolean**.

expression.Dirty

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

Example

This Visual Basic/Visual Basic for Applications example returns the value of the **Dirty** property as the value of a global variable.

vate Property Get PropertyPage_ Dirty () As Bo PropertyPage_Dirty = globDirty Property	olean

DisplayName Property

For **Attachment** object:

Returns or sets a **String** representing the name, which need not be the actual file name, displayed below the icon representing the embedded attachment. This property corresponds to the MAPI property PR_DISPLAY_NAME. Read/write.

For **FormDescription** object:

Returns or sets a **String** representing the name of the form, whish is what will be displayed in the **Choose Forms** dialog box. If both **FormDescription.Name** and **FormDescription.DisplayName** properties are empty, setting one will set the other. If one has been previously set, setting the other will not change the value. Read/write.

expression.DisplayName

expression Required. An expression that returns an **Attachment** or **FormDescription** object.

Example

This Visual Basic for Applications example uses the **SaveAsFile** method to save the first attachment of the currently open item as a file in the My Documents folder, using the attachment's display name as the file name.

```
Set myOlApp = CreateObject("Outlook.Application")
Set myinspector = myOlApp.ActiveInspector
If Not TypeName(myinspector) = "Nothing" Then
    If TypeName(myinspector.CurrentItem) = "MailItem" Then
    Set myitem = myinspector.CurrentItem
    Set myAttachments = myitem.attachments
    myAttachments.Item(1).SaveAsFile "C:\" & _
    myAttachments.Item(1).DisplayName
    End If
    MsgBox "The item is of the wrong type."
End If
```

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

```
Set myinspector = Application.ActiveInspector
If Not TypeName(myinspector) = "Nothing" Then
    If TypeName(myinspector.CurrentItem) = "MailItem" Then
    Set myitem = myinspector.CurrentItem
    Set myAttachments = myitem.attachments
    myAttachments.Item(1).SaveAsFile "C:\" & _
    myAttachments.Item(1).DisplayName
    End If
    MsgBox "The item is of the wrong type."
End If
```

DisplayType Property

Returns an OlDisplayType constant that describes the nature of the recipient. Read-only.
OlDisplayType can be one of these OlDisplayType constants.
olAgent
olDistList
olForum
olOrganization
olPrivateDistList
olRemoteUser
olUser
You can use the DisplayType property to filter recipients. The DisplayType property corresponds to the MAPI property PR_DISPLAY_TYPE.
expression.DisplayType
<i>expression</i> 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

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

Example

This Microsoft Visual Basic/Visual Basic for Applications example creates a new distribution list and then prompts the user for the name.

```
Dim myOlApp As New Outlook.Application
Dim myDistList As Outlook.DistListItem
Set myDistList = myOlApp.CreateItem(olDistributionListItem)
myDistList.DLName = InputBox("Type the name of the new distribution myDistList.Display
```

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

```
Set myDistList = Application.CreateItem(7)
myDistList.DLName = _
    InputBox("Type the name of the new distribution list.")
myDistList.Display
```

▼ Show All

DocPosted Property

True if the journalized <u>item</u> was posted as part of the journalized <u>session</u> . Read/write Boolean .		

▼ Show All

DocPrinted Property

True if the journalized <u>item</u> was printed as part of the journalized <u>session</u> . Read/write Boolean .		
expression.DocPrinted		
expression	Required. An expression that returns a JournalItem object.	

▼ Show All

DocRouted Property

True if the journalized <u>item</u> was routed as part of the journalized <u>session</u> . Read/write Boolean .		
expression.DocRouted		
expression	Required. An expression that returns a JournalItem object.	

▼ Show All

DocSaved Property

True if the journalized <u>item</u> was saved as part of the journalized <u>session</u> . Read/write Boolean .		
expression.DocSaved		
expression	Required. An expression that returns a JournalItem object.	

▼ Show All

DownloadState Property

Returns or sets an **OlDownloadState** constant indicating the download state of the item. Read-only **OlDownloadState**.

OlDownloadState can be one of these OlDownloadState constants. **olFullItem** The entire item has been downloaded. **olHeaderOnly** Only the header has been downloaded.

expression.DownloadState

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

Example

The following example searches through the user's Inbox for items that have not yet been downloaded. If any such items are found, a message is displayed to the user and the item is marked for download.

```
Public Sub DownloadStateTest()
'Tests items in the user's Inbox
    Dim outApp As Outlook.Application
    Dim mpfInbox As Outlook.MAPIFolder
    Dim obj As Object
    Set outApp = CreateObject("Outlook.Application")
    Set mpfInbox = outApp.GetNamespace("MAPI").GetDefaultFolder(olfo
    ' Loop all items in the Inbox Folder
    For i = 1 To mpfInbox.Items.Count
        Set obj = mpfInbox.Items.Item(i)
        'Verify if the state of the item is olHeaderOnly
        If obj.DownloadState = olHeaderOnly Then
            MsgBox ("This item has not been fully downloaded")
            'Mark the item to be downloaded
            obi.MarkForDownload = olMarkedForDownload
        End If
    Next
End Sub
```

DueDate Property

Returns or sets a Date indicating the due date for the task. Read/write.			
expression.DueDate			
expression	Required. An expression that returns a TaskItem object.		

Duration Property

-

Returns or sets a **Long** indicating the duration (in minutes) of the appointment, journal entry, or recurrence pattern. For recurrences, this property is only valid for appointments. Read/write.

expression.Duration

expression Required. An expression that returns an **AppointmentItem**, **RecurrencePattern**, or **JournalItem** object.

Example

This Visual Basic for Applications example uses <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.

```
Set myOlApp = CreateObject("Outlook.Application")
Set myitem = myOlApp.CreateItem(olAppointmentItem)
myitem.MeetingStatus = olMeeting
myitem.Subject = "Strategy Meeting"
myitem.Location = "Conference Room B"
myitem.Start = #9/24/1997 1:30:00 PM#
myitem.Duration = 90
Set myRequiredAttendee = myitem.Recipients.Add("Nate Sun")
myRequiredAttendee.Type = olRequired
Set myOptionalAttendee = myitem.Recipients.Add("Kevin Kennedy")
myOptionalAttendee.Type = olOptional
Set myResourceAttendee = myitem.Recipients.Add("Conference Room B")
myResourceAttendee.Type = olResource
myitem.Display
```

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

```
Set myItem = Application.CreateItem(1)
myItem.MeetingStatus = 1
myItem.Subject = "Strategy Meeting"
myItem.Location = "Conference Room B"
myItem.Start = #9/24/97 1:30:00 PM#
myItem.Duration = 90
Set myRequiredAttendee = myItem.Recipients.Add ("Nate Sun")
myRequiredAttendee.Type = 1
Set myOptionalAttendee = myItem.Recipients.Add ("Kevin Kennedy")
myOptionalAttendee.Type = 2
Set myResourceAttendee = myItem.Recipients.Add("Conference Room B")
myResourceAttendee.Type = 3
myItem.Display
```

		_

EditorType Property

Returns an **OlEditorType** constant indicating the type of editor. Read-only.

OlEditorType can be one of these OlEditorType constants.

olEditorHTML olEditorRTF olEditorText olEditorWord

Note The **EditorType** property is not affected when you merely access the **Body** property of the item (as in MsgBox myItem.Body), but when you reset the **Body** property (as in myItem.Body = "This is a new body"), the **EditorType** reverts back to the user's default editor.

expression.EditorType

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

This VBScript example uses the **Open** event to access the **HTMLBody** property of an item. This sets the **EditorType** property of the item's **Inspector** to **olEditorHTML**. When the item's **Body** property is set, the **EditorType** property is changed to the default. For example, if the default editor is set to RTF, the **EditorType** is set to **olEditorRTF**.

If this code is placed in the Script Editor of a form in design mode, the message boxes during run time will reflect the change in the **EditorType** as the body of the form changes. The final message box utilizes the **ScriptText** property to display all the VBScript code in the Script Editor.

```
Function Item_Open()
    'Set the HTMLBody of the item.
    Item.HTMLBody = "<HTML><H2>My HTML page.</H2><BODY>My body.</BOD</pre>
    '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. Email 1Address

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

This Visual Basic for Applications example sets "someone@microsoft.com" as the e-mail address for the first e-mail entry of a contact.

```
Set myOlApp = CreateObject("Outlook.Application")
Set myItem = myOlApp.CreateItem(olContactItem)
myItem.Email1Address = "someone@microsoft.com"
```

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

```
Set myItem = Application.CreateItem(2)
myItem.Email1Address = "someone@microsoft.com"
```

Email1AddressType Property

Returns or sets a **String** representing the address type (such as EX or SMTP) of the first e-mail entry for the contact. This is a free-form text field, but it must match the actual type of an existing mail transport. Read/write.

expression.Email1AddressType

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

This Visual Basic for Applications example sets "SMTP" as the address type for the first e-mail entry of a contact.

```
Set myOlApp = CreateObject("Outlook.Application")
Set myItem = myOlApp.CreateItem(olContactItem)
myItem.Email1AddressType = "SMTP"
```

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

```
Set myItem = Application.CreateItem(2)
myItem.Email1AddressType = "SMTP"
```

Email1DisplayName Property

	ring representing the display name of the first e-mail address for This property is set to the value of the FullName property by d-only.
expression . E	Email1DisplayName
expression	Required. An expression that returns a ContactItem object.

Email1EntryID Property

Returns a String representing the entry ID of the first e-mail address for the contact. Read-only.		
expression . E	Email1EntryID	
expression	Required. An expression that returns a ContactItem object.	

Email2Address Property

Returns or sets a **String** representing the e-mail address of the second e-mail entry for the contact. Read/write.

expression. Email 2Address

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

This Visual Basic for Applications example sets "someone@microsoft.com" as the e-mail address for the second e-mail entry of a contact.

```
Set myOlApp = CreateObject("Outlook.Application")
Set myItem = myOlApp.CreateItem(olContactItem)
myItem.Email2Address = "someone@microsoft.com"
```

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

```
Set myItem = Application.CreateItem(2)
myItem.Email2Address = "someone@microsoft.com"
```

Email2AddressType Property

Returns or sets a **String** representing the address type (such as EX or SMTP) of the second e-mail entry for the contact. This is a free-form text field, but it must match the actual type of an existing mail transport. Read/write.

expression.Email2AddressType

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

This Visual Basic for Applications example sets "SMTP" as the address type for the second e-mail entry of a contact.

```
Set myOlApp = CreateObject("Outlook.Application")
Set myItem = myOlApp.CreateItem(olContactItem)
myItem.Email2AddressType = "SMTP"
```

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

```
Set myItem = Application.CreateItem(2)
myItem.Email2AddressType = "SMTP"
```

Email2DisplayName Property

	ring representing the display name of the second e-mail entry for This property is set to the value of the FullName property by d-only.
expression . E	Email2DisplayName
expression	Required. An expression that returns a ContactItem object.

Email2EntryID Property

Returns a String representing the entry ID of the second e-mail entry for the contact. Read-only.		
expression.Email2EntryID		
<i>expression</i> Required. An expression that returns a ContactItem object.		

Email3Address Property

Returns or sets a **String** representing the e-mail address of the third e-mail entry for the contact. Read/write.

expression. Email 3Address

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

This Visual Basic for Applications example sets "someone@microsoft.com" as the e-mail address for the third e-mail entry of a contact.

```
Set myOlApp = CreateObject("Outlook.Application")
Set myItem = myOlApp.CreateItem(olContactItem)
myItem.Email3Address = " someone@microsoft.com "
```

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

```
Set myItem = Application.CreateItem(2)
myItem.Email3Address = "someone@microsoft.com"
```

Email3AddressType Property

Returns or sets a **String** representing the address type (such as EX or SMTP) of the third e-mail entry for the contact. This is a free-form text field, but it must match the actual type of an existing mail transport. Read/write.

expression.Email3AddressType

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

This Visual Basic for Applications example sets "SMTP" as the address type for the third e-mail entry of a contact.

```
Set myOlApp = CreateObject("Outlook.Application")
Set myItem = myOlApp.CreateItem(olContactItem)
myItem.Email3AddressType = "SMTP"
```

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

```
Set myItem = Application.CreateItem(2)
myItem.Email3AddressType = "SMTP"
```

Email3DisplayName Property

ring representing the display name of the third e-mail entry for the property is set to the value of the FullName property by default.
mail3DisplayName
Required. An expression that returns a ContactItem object.

Email3EntryID Property

Returns a String representing the entry ID of the third e-mail entry for the contact. Read-only.		
expression.Email3EntryID		
<i>expression</i> Required. An expression that returns a ContactItem object.		

Enabled Property

True if the action is enabled in the application. Read/write Boolean .		
expression.Enabled		
expression	Required. An expression that returns an Action object.	

End Property

_

Returns or sets a **Date** indicating the end date and time of an appointment or journal entry. Read/write.

expression.End

expression Required. An expression that returns an **AppointmentItem** or a **JournalItem** object.

This Visual Basic for Applications example uses <u>CreateItem</u> to create an <u>AppointmentItem</u> object. The <u>RecurrencePattern</u> is obtained for this item using the <u>GetRecurrencePattern</u> method. By setting the <u>RecurrencePattern</u> properties, <u>RecurrenceType</u>, <u>PatternStartDate</u>, and <u>PatternEndDate</u>, the appointments are now a recurring series that occur on a daily basis for the period of one year.

An **Exception** object is created when one instance of this recurring appointment is obtained using the **GetOccurrence** method and properties for this instance are altered. This exception to the series of appointments is obtained using the **GetRecurrencePattern** method to access the **Exceptions** collection associated with this series. Message boxes display the original **Subject** and **OriginalDate** for this exception to the series of appointments and the current date, time, and subject for this exception.

For a description of changes required for this example to work in VBScript, see the Note at the end of the example.

```
Public Sub cmdExample()
    Set myOlApp = New Outlook.Application
    Set myApptItem = myOlApp.CreateItem(olAppointmentItem)
   myApptItem.Start = #2/2/98 3:00:00 PM#
    myApptItem.End = #2/2/98 4:00:00 PM#
   myApptItem.Subject = "Meet with Boss"
    'Get the recurrence pattern for this appointment
    'and set it so that this is a daily appointment
    'that begins on 2/2/98 and ends on 2/2/99
     'and save it.
    Set myRecurrPatt = myApptItem.GetRecurrencePattern
   myRecurrPatt.RecurrenceType = olRecursDaily
   myRecurrPatt.PatternStartDate = #2/2/98#
    myRecurrPatt.PatternEndDate = #2/2/99#
   myApptItem.Save
    'Access the items in the Calendar folder to locate
    'the master AppointmentItem for the new series.
    Set myNamespace = myOlApp.GetNamespace("MAPI")
    Set myFolder = myNamespace.GetDefaultFolder(olFolderCalendar)
```

```
Set myItems = myFolder.Items
    Set myApptItem = myItems("Meet with Boss")
    'Get the recurrence pattern for this appointment
    'and obtain the occurrence for 3/12/98.
    myDate = #3/12/98 \ 3:00:00 \ PM#
    Set myRecurrPatt = myApptItem.GetRecurrencePattern
    Set myOddApptItem = myRecurrPatt.GetOccurrence(myDate)
    'Save the existing subject. Change the subject and
    'starting time for this particular appointment
    'and save it.
    saveSubject = myOddApptItem.Subject
    myOddApptItem.Subject = "Meet NEW Boss"
    newDate = #3/12/98 \ 3:30:00 \ PM#
    myOddApptItem.Start = newDate
    myOddApptItem.Save
    'Get the recurrence pattern for the master
    'AppointmentItem. Access the collection of
    'exceptions to the regular appointments.
    Set myRecurrPatt = myApptItem.GetRecurrencePattern
    Set myException = myRecurrPatt.Exceptions.Item(1)
    'Display the original date, time, and subject
    'for this exception.
    MsgBox myException.OriginalDate & ": " & saveSubject
    'Display the current date, time, and subject
    'for this exception.
    MsgBox myException.AppointmentItem.Start & ": " &
myException.AppointmentItem.Subject
End Sub
```

Note For this example to work properly in VBScript, there are only a few simple changes that need to be made in the code.

You don't have to retrieve the application as an object and you must use the values of the constants, so:

```
Set myOlApp = New Outlook.Application
    Set myApptItem = myOlApp.CreateItem(olAppointmentItem)
becomes:
Set myApptItem = Application.CreateItem(1)
```

```
and
myRecurrPatt.RecurrenceType = olRecursDaily
becomes:
myRecurrPatt.RecurrenceType = 0
and
Set myFolder = myNamespace.GetDefaultFolder(olFolderCalendar)
becomes:
Set myFolder = myNamespace.GetDefaultFolder(9)
```

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 .I	End
expression	Required. An expression that returns a RecurrencePattern object.

▼ Show All

EntryID Property

_

Returns a **String** representing the unique entry ID of the object. This property corresponds to the MAPI property PR_ENTRYID. MAPI systems assign a permanent, unique ID string when an object is created that does not change from one MAPI <u>session</u> to another. The **EntryID** property is not set for an Outlook <u>item</u> until it is saved or sent. Also, the **EntryID** changes when an item is moved into another folder. Read-only.

expression. EntryID

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

This Visual Basic for Applications example uses the **EntryID** property to compare the entry ID of one message to the entry ID of a message returned by a search operation to determine whether the objects represent the same message.

Exceptions Property

Returns the **Exceptions** collection for a specified series of recurring appointments.

expression. Exceptions

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

This Visual Basic for Applications example uses <u>CreateItem</u> to create an <u>AppointmentItem</u> object. The <u>RecurrencePattern</u> is obtained for this item using the <u>GetRecurrencePattern</u> method. By setting the <u>RecurrencePattern</u> properties, <u>RecurrenceType</u>, <u>PatternStartDate</u>, and <u>PatternEndDate</u>, the appointments are now a recurring series that occur on a daily basis for the period of one year.

An **Exception** object is created when one instance of this recurring appointment is obtained using the **GetOccurrence** method and properties for this instance are altered. This exception to the series of appointments is obtained using the **GetRecurrencePattern** method to access the **Exceptions** collection associated with this series. Message boxes display the original **Subject** and **OriginalDate** for this exception to the series of appointments and the current date, time, and subject for this exception.

For a description of changes required for this example to work in VBScript, see the Note at the end of the example.

```
Public Sub cmdExample()
    Set myOlApp = New Outlook.Application
    Set myApptItem = myOlApp.CreateItem(olAppointmentItem)
   myApptItem.Start = #2/2/98 3:00:00 PM#
    myApptItem.End = #2/2/98 4:00:00 PM#
   myApptItem.Subject = "Meet with Boss"
    'Get the recurrence pattern for this appointment
    'and set it so that this is a daily appointment
    'that begins on 2/2/98 and ends on 2/2/99
     'and save it.
    Set myRecurrPatt = myApptItem.GetRecurrencePattern
   myRecurrPatt.RecurrenceType = olRecursDaily
   myRecurrPatt.PatternStartDate = #2/2/98#
    myRecurrPatt.PatternEndDate = #2/2/99#
   myApptItem.Save
    'Access the items in the Calendar folder to locate
    'the master AppointmentItem for the new series.
    Set myNamespace = myOlApp.GetNamespace("MAPI")
    Set myFolder = myNamespace.GetDefaultFolder(olFolderCalendar)
```

```
Set myItems = myFolder.Items
    Set myApptItem = myItems("Meet with Boss")
    'Get the recurrence pattern for this appointment
    'and obtain the occurrence for 3/12/98.
    myDate = #3/12/98 \ 3:00:00 \ PM#
    Set myRecurrPatt = myApptItem.GetRecurrencePattern
    Set myOddApptItem = myRecurrPatt.GetOccurrence(myDate)
    'Save the existing subject. Change the subject and
    'starting time for this particular appointment
    'and save it.
    saveSubject = myOddApptItem.Subject
    myOddApptItem.Subject = "Meet NEW Boss"
    newDate = #3/12/98 \ 3:30:00 \ PM#
    myOddApptItem.Start = newDate
    myOddApptItem.Save
    'Get the recurrence pattern for the master
    'AppointmentItem. Access the collection of
    'exceptions to the regular appointments.
    Set myRecurrPatt = myApptItem.GetRecurrencePattern
    Set myException = myRecurrPatt.Exceptions.Item(1)
    'Display the original date, time, and subject
    'for this exception.
    MsgBox myException.OriginalDate & ": " & saveSubject
    'Display the current date, time, and subject
    'for this exception.
    MsgBox myException.AppointmentItem.Start & ": " & _
myException.AppointmentItem.Subject
End Sub
```

Note For this example to work properly in VBScript, there are only a few simple changes that need to be made in the code.

You don't have to retrieve the application as an object and you must use the values of the constants, so:

```
Set myOlApp = New Outlook.Application
    Set myApptItem = myOlApp.CreateItem(olAppointmentItem)
becomes:
Set myApptItem = Application.CreateItem(1)
```

```
and

myRecurrPatt.RecurrenceType = olRecursDaily

becomes:

myRecurrPatt.RecurrenceType = 0

and

Set myFolder = myNamespace.GetDefaultFolder(olFolderCalendar)

becomes:

Set myFolder = myNamespace.GetDefaultFolder
```

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 VBScript example uses the <u>Send</u> event and sends an item with an automatic expiration date of one week.

```
Function Item_Send()
   Item.ExpiryTime = Date + 7
End Function
```

Explorers Property

Returns an **Explorers** collection object that contains the **Explorer** objects representing all open explorers.

expression. Explorers

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

The following Microsoft Visual Basic example displays the number of explorer windows that are open.

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

```
Sub CommandButton1_Click()
   MsgBox "There are " & _
        Application.Explorers.Count & " Explorers."

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

FileName Property

Returns a String representing the file name of the attachment. Use this property in conjunction with the PathName property. Read-only.	
expression.FileName	
<i>expression</i> Required. An expression that returns an <u>Attachment</u> object.	

Filter Property

-

The DASL statement used to restrict the search to a specified subset of data. This property is set by the <u>Application</u> object's <u>AdvancedSearch</u> method. Read-only **String**.

expression.Filter

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

Remarks

The **Filter** property is set by the *Filter* argument when the **Search** object is first created.

The following example creates a new **Search** object. The event subroutine fires after the search has finished and displays the **Tag** and **Filter** properties of the **Search** object as well as the results of the search.

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

End Sub
```

FirstName Property

Returns or sets a **String** representing the first name for the contact. Read/write.

expression.FirstName

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

Remarks

This property is parsed from the FullName property, but may be changed or entered independently should it be parsed incorrectly. Note that any such changes or entries to this property will be overwritten by any subsequent
changes of entries to FullName .

FlagDueBy Property

Returns or sets a Date specifying the date by which this mail message is due. This property is only valid if the FlagStatus property is also set for the message. This property corresponds to the MAPI property PR_REPLY_TIME. Read/write
expression.FlagDueBy
<i>expression</i> Required. An expression that returns one of the objects in the Applies To list.

FlagRequest Property

Returns or sets a **String** indicating the requested action for the mail message. This is a free-form text field. This property is only valid if the **FlagStatus** property is also set for the message. Read/write.

expression.FlagRequest**

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

FlagStatus Property

Returns or sets an OlFlagStatus constant indicating the flag status for the mail message. Read/write.

OlFlagStatus can be one of these OlFlagStatus constants.

olFlagComplete
olFlagMarked
olNoFlag

expression.FlagStatus

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

FolderPath Property

Returns a read-only **String** that indicates the path of the current folder.

expression. FolderPath

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

The following example displays information about the current folder. The subroutine accepts a **MAPIFolder** object and displays the folder's name, path, and address book information.

```
Sub Folderpaths()
    Dim olApp As Outlook.Application
    Dim nmsName As NameSpace
    Dim fldFolder As MAPIFolder
    Set olApp = Outlook.Application
    'Create namespace reference
    Set nmsName = olApp.GetNamespace("MAPI")
    'create folder instance
    Set fldFolder = nmsName.GetDefaultFolder(olFolderInbox)
    'call sub program
    Call FolderInfo(fldFolder)
End Sub
Sub FolderInfo(ByVal fldFolder As MAPIFolder)
'Displays information about a given folder
    MsgBox fldFolder.Name & "'s current path is " & fldFolder.Folder
             ". The current address book name is " & fldFolder.Addre
End Sub
```

Folders Property

Returns the <u>Folders</u> collection that represents all the folders contained in the specified folder or name space. The <u>NameSpace</u> 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 example uses the <u>Add</u> method to add the new folder named "My Contacts" to the current (default) Contacts folder.

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

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

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

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

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

▼ Show All

FormDescription Property

Returns the FormDescription object that represents the form description for the specified Microsoft Outlook <u>item</u> .
expression.FormDescription
<i>expression</i> Required. An expression that returns one of the objects listed in the Applies To list.

Formula Property

Returns or se Read/write.	ets a String representing the formula for the user property.
expression .F	ormula
<i>expression</i> Applies To li	Required. An expression that returns one of the objects in the st.

FTPSite Property

Returns or se Read/write.	ets a String representing the FTP site entry for the contact.
expression .F	TPSite
expression	Required. An expression that returns a ContactItem object.

FullName Property

Returns or sets a **String** specifying the whole, unparsed full name for the contact. Read/write.

expression.FullName

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

Remarks

This property is parsed into the **FirstName**, **LastName**, **MiddleName**, and **Suffix** properties, which may be changed or entered independently should they be parsed incorrectly. Note that any such changes or entries to the **FirstName**, **LastName**, **MiddleName** or **Suffix** properties will be overwritten by any subsequent changes or entries to **FullName**.

This Visual Basic for Applications example uses the **Restrict** method to get all Inbox items dealing with Project X and moves them to the Project X folder.

```
Set myOlApp = CreateObject("Outlook.Application")
Set myNameSpace = myOlApp.GetNameSpace("MAPI")
Set myFolder = _
         myNameSpace.GetDefaultFolder(olFolderInbox)
Set myItems = myFolder.Items
Set myRestrictItems = myItems.Restrict _ ("[Categories] = 'Proje
For Each myItem In myRestrictItems
         myItem.Move myFolder.Folders("Project X")
Next
```

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

```
Set myNameSpace = Application.GetNameSpace("MAPI")
Set myFolder = myNameSpace.GetDefaultFolder(6)
Set myItems = myFolder.Items
Set myRestrictItems = myItems.Restrict _ ("[Categories] = 'Proje
For Each myItem In myRestrictItems
    myItem.Move myFolder.Folders("Project X")
Next
```

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

The following Visual Basic for Applications example is the same as the example above, except that it demonstrates the use of a variable in the filter.

```
Public Sub ContactDateCheck()
   Set myOlApp = CreateObject("Outlook.Application")
   Set myNameSpace = myOlApp.GetNamespace("MAPI")
   Set myContacts = myNameSpace.GetDefaultFolder(olFolderContacts).I
   DateStart = #6/11/97#
   DateToCheck$ = "[LastModificationTime] >= """ & DateStart & """"
   Set myRestrictItems = myContacts.Restrict(DateToCheck$)
   For Each myItem In myRestrictItems
        MsgBox myItem.FullName & ": " & MyItem.LastModificationTime
   Next
End Sub
```

FullNameAndCompany Property

Returns a String representing the full name and company of the contact by concatenating the values of the FullName and CompanyName properties. Read-only.	
expression.FullNameAndCompany	
<i>expression</i> Required. An expression that returns a ContactItem object.	

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

▼ Show All

GetInspector Property

-

Returns an **Inspector** object that represents an inspector initialized to contain the specified <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.

Remarks

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

This Visual Basic for Applications example uses the **GetInspector** property to return a new, inactive inspector for myItem, and then displays it.

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

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

```
Set myItem = Application.CreateItem(0)
Set myInspector = myItem.GetInspector
myInspector.Display
```

GovernmentIDNumber Property

Returns or se Read/write.	ts a String representing the government ID number for the contact.
expression.GovernmentIDNumber	
expression	Required. An expression that returns a ContactItem object.

▼ Show All

Groups Property

_

Returns an **OutlookBarGroups** object representing the set of groups in an **Outlook Bar**.

expression. Groups

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

The following Microsoft Visual Basic/Visual Basic for Applications example displays the number of groups in the Outlook Bar.

```
Dim myOlApp As New Outlook.Application
Dim myOlBar As Outlook.OutlookBarPane
Set myOlBar = myOlApp.ActiveExplorer.Panes.Item("OutlookBar")
myCount = myOlBar.Contents.Groups.Count
MsgBox "There are " & myCount & " groups in the Outlook Bar"
```

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

```
Set myOlBar = Application.ActiveExplorer.Panes.Item("OutlookBar")
myCount = myOlBar.Contents.Groups.Count
MsgBox "There are " & myCount & " groups in the Outlook Bar"
```

HasAttachment Property

True (defau Boolean .	lt) if the remote item has an attachment associated with it. Read-only
expression. I	HasAttachment
expression	Required. An expression that returns a RemoteItem object.

Height Property

	ets a Long specifying the height (in pixels) of the explorer, note window. Read/write.
expression .I	Height
<i>expression</i> NoteItem o	Required. An expression that returns an Explorer , Inspector , or bject.

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 .	
Description object.	

Hobby Property

Returns or sets a String representing the hobby for the contact. Read/write.	
Hobby	
Required. An expression that returns a ContactItem object.	

Home2TelephoneNumber Property

Returns or sets a String representing the second home telephone number for the contact. Read/write.	
expression .Hon	ne2TelephoneNumber
expression Re	quired. An expression that returns a ContactItem object.

HomeAddress Property

Returns or sets a String representing the full, unparsed text of the home address for the contact. Read/write.	
expression. H	IomeAddress
expression	Required. An expression that returns a ContactItem object.

HomeAddressCity Property

Returns or sets a **String** representing the city portion of the home address for the contact. Read/write.

expression. HomeAddressCity

This property is parsed from the HomeAddress property, but may be changed centered independently should it be parsed incorrectly. Note that any such	r
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 HomeAddress property, but may be changed centered independently should it be parsed incorrectly. Note that any such	r
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 HomeAddress property, but may be changed centered independently should it be parsed incorrectly. Note that any such	r
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 HomeAddress property, but may be changed centered independently should it be parsed incorrectly. Note that any such	r
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 HomeAddress property, but may be changed centered independently should it be parsed incorrectly. Note that any such	r
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 HomeAddress property, but may be changed centered independently should it be parsed incorrectly. Note that any such	r
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 .Ho	meFaxNumber
expression R	Required. An expression that returns a ContactItem object.

HomeTelephoneNumber Property

Returns or sets a String representing the first home telephone number for the contact. Read/write.				
expression.HomeTelephoneNumber				
expression	Required. An expression that returns a ContactItem object.			

▼ Show All

HTMLBody Property

Returns or sets a **String** representing the HTML body of the specified <u>item</u>. The **HTMLBody** property should be an HTML syntax string. Read/write.

Setting the **HTMLBody** property sets the **EditorType** property of the item's **Inspector** to **olEditorHTML**.

Setting the **HTMLBody** property will always update the **Body** property immediately.

Setting the **Body** property will clear the contents of the **HTMLBody** property on HTML aware stores.

Note The **EditorType** property is not affected when you merely access the **Body** property of the item (as in MsgBox myItem.Body), but when you reset the **Body** property (as in myItem.Body = "This is a new body"), the **EditorType** reverts back to the user's default editor.

expression.HTMLBody

expression Required. An expression that returns a **PostItem** or **MailItem** object.

Example

This VBScript example uses the **Open** event to access the **HTMLBody** property of an **item**. This sets the **EditorType** property of the item's **Inspector** to **olEditorHTML**. When the item's **Body** property is set, the **EditorType** property is changed to the default. For example, if the default editor is set to RTF, the **EditorType** is set to **olEditorRTF**.

If this code is placed in the Script Editor of a form in design time, the message boxes during run time will reflect the change in the **EditorType** as the body of the form changes. The final message box utilizes the **ScriptText** property to display all the VBScript code in the Script Editor.

```
Function Item_Open()
    'Set the HTMLBody of the item.
    Item.HTMLBody = "<HTML><H2>My HTML page.</H2><BODY>My body.</BOD</pre>
    '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
```

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 you must set the homepage property of your **MAPIfolder** to a Web page, or set the **WebViewURL** property of the **MAPIFolder** object to a web page.

Example

The following example accesses the Outlook View Control.

```
Sub GetHTML()
'Returns the Outlook View control

Dim objExp As Explorer
Set objExp = Application.ActiveExplorer
Dim HTMLDoc As HTMLDocument
Set HTMLDoc = objExp.HTMLDocument
Dim objVC As ViewControl
Set objVC = HTMLDoc.All.Tags("object").Item(0).Object

End Sub
```

HTMLEditor Property

Returns an **Object** representing the HTML Document Object Model of the message being displayed. The HTML Document Object Model is defined by Microsoft Internet Explorer and is the same one used for Dynamic HTML. This object may be temporary and should not be stored for later use. Read-only.

The **HTMLEditor** property is only valid if the **EditorType** property of the item's associated **Inspector** is set to **olEditorHTML**.

expression.HTMLEditor

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

Example

The following VBScript example uses the **Click** event of a **CommandButton** control named "CommandButton1" to demonstrate the listing of all HTML elements.

```
Sub CommandButton1_Click()
   Dim i
                 'As Integer
   Dim strHTMLType 'As String
   Dim strHTMLText
                     'As String
   Dim NL 'As String
   NL = chr(10) \& chr(13)
   Set myInspector = Item.GetInspector
   Set myIExplorer = myInspector.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.			
xpression.Icon			
<i>xpression</i> Required. An expression that returns a FormDescription object.			

▼ Show All

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		
<i>expression</i> Required. An expression that returns an AddressEntry or AddressList object.		

IMAddress Property

Returns or sets a **String** that represents a contact's Microsoft Instant Messenger address. Read/write.

expression.IMAddress

Unlike the **Recipients** or **To** properties, there is no way to verify that the **IMAddress** property contains a valid address.

Example

The following example creates a new contact and prompts the user to enter an Instant Messenger address for the contact.

▼ Show All

Importance Property

Returns or sets an Olimportance constant indicating the relative importance level for the Outlook item. This property corresponds to the MAPI property PR_IMPORTANCE. Read/write.

Olimportance can be one of these Olimportance constants.
olimportanceHigh
olimportanceLow
olimportanceNormal

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

InAppFolderSyncObject Property

Returns or sets a **Boolean** that determines if the specified folder will be synchronized with the mail server. If **True**, this folder will be synchronized when the "application folders" **SyncObject** is synchronized. If **False**, the folder will not synchronize. Read/write.

expression.InAppFolderSyncObject

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

This is equivalent to clicking the check box for this folder in the "application folders" group of the **Send/Receive** dialog box.

If this property is set to **True**, and the "application folders" **SyncObject** does not already exist, a **SyncObject** will be automatically created.

The following example sets the current **MAPIFolder** to synchronize when the "Application Folders" **SyncObject** object is synchronized. Here, the **InAppFolderSyncObject** property is used in conjunction with the **AppFolders** property of the **SyncObject**.

```
Public Sub appfolders()
    Dim olApp As New Outlook.Application
    Dim nsp As Outlook.NameSpace
    Dim sycs As Outlook.SyncObjects
    Dim syc As Outlook.SyncObject
    Dim mpfInbox As Outlook.MAPIFolder
    Set nsp = olApp.GetNamespace("MAPI")
    Set sycs = nsp.SyncObjects
    'Return the application folder SyncObject
    Set syc = sycs.appfolders
    'Get the Inbox folder
    Set mpfInbox = nsp.GetDefaultFolder(olFolderInbox)
    'Set the Inbox Folder to be synchronized when the application
    'folder's SyncObject is synchronized
   mpfInbox.InAppFolderSyncObject = True
    'Star the synchronization
    syc.Start
End Sub
```

IncludeRecurrences Property

True if the <u>Items</u> collection should include recurrence patterns. This property only has an effect if the <u>Items</u> collection contains appointments and is not sorted by any property other than <u>Start</u> in ascending order. The default value is <u>False</u>. Read/write <u>Boolean</u>.

expression.IncludeRecurrences

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

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.

▼ Show All

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 **AddressList**, **Attachment**, or **Recipient** object.

Initials Property

Returns or sets a String representing the initials for the contact. Read/write.		
expression.	Initials	
expression	Required. An expression that returns a ContactItem object.	

Inspectors Property

Returns an **Inspectors** collection object that contains the **Inspector** objects representing all open inspectors.

expression. Inspectors

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

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.

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

```
Private Sub Command1_Click()
   If Application.Inspectors.Count > 0 Then
        For x = 1 To myOlApp.Inspectors.Count
             MsgBox Inspectors.Item(x).Caption
        Next x
   Else
        MsgBox "There are no inspector windows open."
   End If
End Sub
```

Instance Property

Returns or sets a **Long** specifying the count for which the recurrence pattern is valid for a given interval. This property is only valid for recurrences of the **olRecursMonthNth** and **olRecursYearNth** type and allows the definition of a recurrence pattern that is only valid for the Nth occurrence, such as "the 2nd Sunday in March" pattern. The count is set numerically: 1 for the first, 2 for the second, and so on through 5 for the last. Values greater than 5 will generate errors when the pattern is saved. Read/write.

expression.Instance		
expression	Required. An expression that returns a RecurrencePattern object.	

InternetCodepage Property

Returns or sets a **Long** that determines the Internet code page used by the item. The Internet code page defines the text encoding scheme used by the item. Read/write.

expression. Internet Codepage

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

The following example searches through the user's Inbox and displays the sender names for all mail items with the Internet code page value 1256. This value corresponds to the Internet code page value for Arabic text. If no mail items are found, a message is displayed to the user.

```
Sub FindArabicUsers()
'Displays information about an Internet code page
    Dim olApp As Outlook.Application
    Dim objMail As MailItem
    Dim objItems As Items
    Dim strUsers As String
    Const cstArabic As String = "1256"
    Set olApp = Outlook.Application
    Set objItems = _
        olApp.GetNamespace("MAPI").GetDefaultFolder(olFolderInbox).I
    For Each objMail In objItems
        If objMail.InternetCodepage = cstArabic Then
            strUser = strUser & objMail.SenderName & vbCr
        End If
    Next objMail
    If strUsers = "" Then
        MsgBox "There are no Mail items in the Inbox from users with
            & "Internet code page."
    Else
        MsgBox "The following users use the specified foreign Intern
                & vbCr & strUser
    End If
End Sub
```

InternetFreeBusyAddress Property

Returns or sets a String corresponding to the Address box on the Details tab for a contact. This box can contain the URL location of the user's free-busy information in vCard Free-Busy standard format. Read/write.	
expression.InternetFreeBusyAddress	
<i>expression</i> Required. An expression that returns a ContactItem object.	

Interval Property

Returns or sets a **Long** specifying the number of units of a given recurrence type between occurrences. For example, setting the **Interval** property to 2 and the **RecurrenceType** property to "Weekly" would cause the pattern to occur every

Note The **Interval** property must be set before setting **PatternEndDate**. Also, the **Interval** property is not valid for yearly recurrence patterns.

expression.Interval

second week. Read/write.

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

This Visual Basic for Applications example uses <u>GetRecurrencePattern</u> to obtain the <u>RecurrencePattern</u> object for the newly-created <u>AppointmentItem</u>. The properties, <u>RecurrenceType</u>, <u>DayOfWeekMask</u>, <u>PatternStartDate</u>, <u>Interval</u>, <u>PatternEndDate</u>, and <u>Subject</u> are set, the appointment is saved and then displayed with the pattern: "Occurs every 3 week(s) on Monday effective 1/21/98 until 12/21/2001 from 2:00 PM to 5:00 PM."

```
Set myOlApp = New Outlook.Application
Set myApptItem = myOlApp.CreateItem(olAppointmentItem)
Set myRecurrPatt = myApptItem.GetRecurrencePattern

myRecurrPatt.RecurrenceType = olRecursWeekly
myRecurrPatt.DayOfWeekMask = olMonday
myRecurrPatt.PatternStartDate = #1/21/98 2:00:00 PM#
myRecurrPatt.Interval = 3
myRecurrPatt.PatternEndDate = #12/21/2001 5:00:00 PM#
myApptItem.Subject = "Important Appointment"
myApptItem.Save
myApptItem.Display
```

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

```
Set myApptItem = Application.CreateItem(1)
Set myRecurrPatt = myApptItem.GetRecurrencePattern
myRecurrPatt.RecurrenceType = 1
myRecurrPatt.DayOfWeekMask = 2
myRecurrPatt.PatternStartDate = #1/21/98 2:00:00 PM#
myRecurrPatt.Interval = 3
myRecurrPatt.PatternEndDate = #12/21/2001 5:00:00 PM#
myApptItem.Subject = "Important Appointment"
myApptItem.Save
myApptItem.Display
```

IsConflict Property

Returns a **Boolean** that determines if the item is in conflict. Whether or not an item is in conflict is determined by the state of the application. For example, when a user is offline and tries to access an online folder the action will fail. In this scenario, the **IsConflict** property will return **True**. Read-only.

expression.IsConflict

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

If **True**, the specified item is in conflict.

The following example creates a new message and attempts to send it. If the **IsConflict** property returns **True**, the item will not be sent.

```
Sub NewMail()
'Creates and attempts to send a new e-mail message

Dim olApp As Outlook.Application
Dim objNewMail As MailItem

Set olApp = Outlook.Application
Set objNewMail = olApp.CreateItem(olMailItem)
objNewMail.Body = "This e-mail message was created automatically objNewMail.To = "Jeff Smith"

If objNewMail.IsConflict = False Then
objNewMail.Send
Else
MsgBox "Conflict: Cannot send e-mail"
End If

End Sub
```

ISDNNumber Property

Returns or sets a String representing the ISDN number for the contact. Read/write.		
expression .I	SDNNumber	
expression	Required. An expression that returns a ContactItem object.	

IsOnlineMeeting Property

True if this is an online meeting. Read/write Boolean .		
expression.IsOnlineMeeting		
expression	Required. An expression that returns an AppointmentItem object.	

IsReadOnly Property

Indicates that the **AddressList** object cannot be modified. The **IsReadOnly** property refers to adding and deleting the entries in the address book container represented by the **AddressList** object. The property is **True** if no entries can be added or deleted. The property is **False** if the container can be modified, that is, if address entries can be added to and deleted from the container.

The **IsReadOnly** property refers to the address book entries in the context of the address book container. It does not indicate whether the contents of the individual entries themselves can be modified.

Read-only l	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 GetRecurrencePattern method is used with an AppointmentItem or TaskIten object, this property is set to True . Read-only Boolean .
expression.IsRecurring
<i>expression</i> Required. An expression that returns a TaskItem or an AppointmentItem object.

IsSynchronous Property

Returns a **Boolean** indicating whether the search is synchronous.

expression. Is Synchronous

expression Required. An expression that returns a **Search** 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 Search object's
Stop method to halt the search.

In order to get meaningful results from an asynchronous search, use the AdvancedSearchComplete event to notify you when the search has finished.

IsUserProperty Property

Returns a **Boolean** value that indicates if the item property is a custom property created by the user. Read-only.

expression. Is User Property

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
           MsgBox "The property " & objitems(i).Name & " was created
        End If
    Next i
End Sub
```

IsVisible Property

Returns a **Boolean** that determines if the reminder is currently visible. All active reminders are visible. If **True**, the reminder is visible. Read-only.

expression.IsVisible

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

This property is determined when it is returned based on the state of the current reminder.

The following example dismisses all reminders that are currently visible. For example, if the current reminder is active, the **IsVisible** property will return **True**.

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 <u>ItemProperties</u> collection that represents all properties associated with an item.

expression. ItemProperties

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

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 example returns the **ItemProperties** collection associated with a **MailItem** object.

```
Sub ItemProperty()
'Creates a new mail item and accesses its properties

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

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

End Sub
```

Items Property

_

Returns an <u>Items</u> collection as a collection of Outlook items in the specified folder.

expression.Items

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

Example

This Visual Basic for Applications example uses the <u>Items</u> property to obtain the collection of <u>MailItem</u> objects from the default Inbox folder.

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

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

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

JobTitle Property

Returns or sets a String representing the job title for the contact. Read/write.			
expression.JobTitle			
expression	Required. An expression that returns a ContactItem object.		

Journal Property

True if the transaction of the contact will be journalized. The default value is False . Read/write Boolean .			
expression.Journal			

Language Property

Returns or sets the language setting for the object that defines the language used in the menu. The **Language** property uses a **String** to represent an ISO language tag. For example, the string ""EN-US" represents the ISO code for "United States - English". Read/write.

expression.Language

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

Remarks

If a valid language code is specified, the object will only be available in the **View** menu for the specified language type. If no value is specified, the object item is available for all language types. The default value for this property is an empty **String**.

Example

The following example sets the language type of all **View** object's of type **olTableView** to US English.

```
Sub SetLanguage()
'Sets the language of all table views to US English
    Dim olApp As Outlook.Application
    Dim objViews As Views
    Dim objView As View
    Set olApp = Outlook.Application
    Set objViews = _
        olApp.GetNamespace("Mapi").GetDefaultFolder(olFolderInbox).V
    'Iterate through each view in the collection
    For Each objView In objViews
        'If view is of type olTableVIew then set language
        If objView.ViewType = olTableView Then
            objView.Language = "EN-US"
        End If
    Next objView
End Sub
```

LanguageSettings Property

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. Last First And Suffix

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



LastFirstNoSpaceAndSuffix Property

Returns the last name, first name, and suffix of the user without a space. Readonly **String**.

expression. LastFirstNoSpaceAndSuffix

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

Remarks

This property is used only when the <code>FirstName</code> , <code>LastName</code> , and <code>Suffix</code> properties (the fields that define this property) contain Asian (DBCS) characters. Note that any such changes or entries to the <code>FirstName</code> , <code>LastName</code> , or <code>Suffix</code> properties will be overwritten by any subsequent changes or entries to <code>FullName</code> .

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 **ContactItem** 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 **ContactItem** object.

ı			
Т			
Т			
Т			
ı			
ı			
ı			
ı			
ı			
ı			
Т			
ı			
1			
-			

LastFirstSpaceOnlyCompany Property

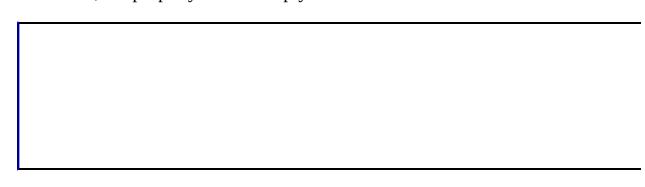
_

Returns a **String** representing the concatenated last name, first name, and middle name of the contact with spaces between them. The company name for the contact is after the middle name. This property is parsed from the **LastName**, **FirstName**, **MiddleName**, and **CompanyName** properties. Read-only.

Note The **LastName**, **FirstName**, and **MiddleName** properties are themselves parsed from the **FullName** property.

expression.LastFirstSpaceOnlyCompany

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



▼ Show All

LastModificationTime Property

Returns a **Date** specifying the 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. Last Modification Time

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

Example

This Visual Basic for Applications example uses the **Restrict** method to get all Inbox items dealing with Project X and moves them to the Project X folder.

```
Set myOlApp = CreateObject("Outlook.Application")
Set myNameSpace = myOlApp.GetNameSpace("MAPI")
Set myFolder = _
         myNameSpace.GetDefaultFolder(olFolderInbox)
Set myItems = myFolder.Items
Set myRestrictItems = myItems.Restrict _ ("[Categories] = 'Proje
For Each myItem In myRestrictItems
         myItem.Move myFolder.Folders("Project X")
Next
```

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

```
Set myNameSpace = Application.GetNameSpace("MAPI")
Set myFolder = myNameSpace.GetDefaultFolder(6)
Set myItems = myFolder.Items
Set myRestrictItems = myItems.Restrict _ ("[Categories] = 'Proje
For Each myItem In myRestrictItems
    myItem.Move myFolder.Folders("Project X")
Next
```

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

The following Visual Basic for Applications example is the same as the example above, except that it demonstrates the use of a variable in the filter.

```
Public Sub ContactDateCheck()
   Set myOlApp = CreateObject("Outlook.Application")
   Set myNameSpace = myOlApp.GetNamespace("MAPI")
   Set myContacts = myNameSpace.GetDefaultFolder(olFolderContacts).I
   DateStart = #6/11/97#
   DateToCheck$ = "[LastModificationTime] >= """ & DateStart & """"
   Set myRestrictItems = myContacts.Restrict(DateToCheck$)
   For Each myItem In myRestrictItems
        MsgBox myItem.FullName & ": " & MyItem.LastModificationTime
   Next
End Sub
```

LastName Property

Returns or sets a **String** representing the last name for the contact. Read/write.

expression.LastName

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

Remarks

This property is parsed from the FullName property, but may be changed or entered independently should it be parsed incorrectly. Note that any such changes or entries to this property will be overwritten by any subsequent		
changes of entries to FullName .		

LastNameAndFirstName Property

Returns a **String** representing the concatenated last name and first name for the contact. Read-only.

expression. LastNameAndFirstName

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

Remarks

This property is parsed from the <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
<i>expression</i> Required. An expression that returns an Explorer , Inspector , or NoteItem object.

Links Property

_

Returns a <u>Links</u> collection of <u>Link</u> objects that represent the contacts to which the item is linked.

expression.Links

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

Example

This Microsoft Visual Basic/Visual Basic for Applications example steps through the items in the Tasks folder and, if a task is not complete, displays the number of contacts linked to the item.

```
Dim myOlApp As New Outlook.Application
Dim myNSpace As Outlook.NameSpace
Dim myItems As Outlook. Items
Dim myItem As Outlook.TaskItem
Dim myLinks As Outlook.Links
Dim myLink As Outlook.Link
Set myNSpace = myOlApp.GetNamespace("MAPI")
Set myItems = myNSpace.GetDefaultFolder(olFolderTasks).Items
For x = 1 To myItems.Count
    If TypeName(myItems.Item(x)) = "TaskItem" Then
        Set myItem = myItems.Item(x)
        Set myLinks = myItem.Links
        Msg = myItem.Subject & " has " & myLinks.Count & " links."
        If myItem.Complete = False Then
            If MsgBox(Msg, vbOKCancel) = vbCancel Then Exit For
        End If
    End If
Next x
```

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

Location Property

Returns or sets a **String** representing the specific office location (for example, Building 1 Room 1 or Suite 123) for the appointment. This property corresponds to the MAPI property PR_OFFICE_LOCATION. Read/write.

expression.Location

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

Example

This Visual Basic for Applications example uses <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.

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

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

```
Set myItem = Application.CreateItem(1)
myItem.MeetingStatus = 1
myItem.Subject = "Strategy Meeting"
myItem.Location = "Conference Room B"
myItem.Start = #9/24/97 1:30:00 PM#
myItem.Duration = 90
Set myRequiredAttendee = myItem.Recipients.Add ("Nate Sun")
myRequiredAttendee.Type = 1
Set myOptionalAttendee = myItem.Recipients.Add ("Kevin Kennedy")
myOptionalAttendee.Type = 2
Set myResourceAttendee = myItem.Recipients.Add("Conference Room B")
myResourceAttendee.Type = 3
myItem.Send
```

Locked Property

True if the form cannot be modified. Read/write Boolean .				
expression.Locked				
expression	Required. An expression that returns a FormDescription object.			

LockUserChanges Property

Returns or sets a value that indicates whether a user can modify the settings of the current view. Read/write **Boolean**.

expression. LockUserChanges

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

Remarks

If **True**, the user can modify the settings of the current view; however, changes made to the interface will not be saved. If **False** (the default), any changes will be saved.

Example

End Sub

The following example locks the user interface for all views that are available to all users. The subroutine LockView accepts the **View** object and a **Boolean** value that indicates if the View interface will be locked. In this example the procedure is always called with the **Boolean** value set to **True**.

```
Sub LocksPublicViews()
'Locks the interface of all views that are available to
'all users of this folder
    Dim olApp As Outlook.Application
    Dim objName As NameSpace
    Dim objViews As Views
    Dim objView As View
    Set olApp = Outlook.Application
    Set objName = olApp.GetNamespace("MAPI")
    Set objViews = objName.GetDefaultFolder(olFolderNotes).Views
    For Each objView In objViews
        If objView.SaveOption = olViewSaveOptionThisFolderEveryone 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
```

MailingAddress Property

Returns or sets a String representing the full, unparsed selected mailing address for the contact. Read/write.

expression. MailingAddress

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

This property replicates the property indicated by the SelectedMailingAddress
property, which is one of the following OlMailingAddress constants:
olBusiness , olHome , olNone , or olOther . While it can be changed or entered
independently, any such changes or entries to this property will be overwritten
by any subsequent changes or entries to the property indicated by
SelectedMailingAddress.

MailingAddressCity Property

Returns or sets a **String** representing the city name portion of the selected mailing address of the contact. Read/write.

expression. MailingAddressCity

This property replicates the property indicated by the SelectedMailingAddress
property, which is one of the following OlMailingAddress constants:
olBusiness , olHome , olNone , or olOther . While it can be changed or entered
independently, any such changes or entries to this property will be overwritten
by any subsequent changes or entries to the property indicated by
SelectedMailingAddress.

MailingAddressCountry Property

Returns or sets a **String** representing the country/region code portion of the selected mailing address of the contact. Read/write.

expression.MailingAddressCountry

This property replicates the property indicated by the SelectedMailingAddress
property, which is one of the following OlMailingAddress constants:
olBusiness , olHome , olNone , or olOther . While it can be changed or entered
independently, any such changes or entries to this property will be overwritten
by any subsequent changes or entries to the property indicated by
SelectedMailingAddress.

MailingAddressPostalCode Property

Returns or sets a **String** representing the postal code (zip code) portion of the selected mailing address of the contact. Read/write.

expression. Mailing AddressPostalCode

This property replicates the property indicated by the SelectedMailingAddress
property, which is one of the following OlMailingAddress constants:
olBusiness , olHome , olNone , or olOther . While it can be changed or entered
independently, any such changes or entries to this property will be overwritten
by any subsequent changes or entries to the property indicated by
SelectedMailingAddress.

MailingAddressPostOfficeBox Property

Returns or sets a **String** representing the post office box number portion of the selected mailing address of the contact. Read/write.

expression. Mailing AddressPostOfficeBox

This property replicates the property indicated by the SelectedMailingAddress
property, which is one of the following OlMailingAddress constants:
olBusiness , olHome , olNone , or olOther . While it can be changed or entered
independently, any such changes or entries to this property will be overwritten
by any subsequent changes or entries to the property indicated by
SelectedMailingAddress.

MailingAddressState Property

Returns or sets a **String** representing the state code portion for the selected mailing address of the contact. Read/write.

expression. Mailing AddressState

This property replicates the property indicated by the SelectedMailingAddress
property, which is one of the following OlMailingAddress constants:
olBusiness , olHome , olNone , or olOther . While it can be changed or entered
independently, any such changes or entries to this property will be overwritten
by any subsequent changes or entries to the property indicated by
SelectedMailingAddress.

MailingAddressStreet Property

Returns or sets a **String** representing the street address portion of the selected mailing address of the contact. Read/write.

expression.MailingAddressStreet

This property replicates the property indicated by the SelectedMailingAddress
property, which is one of the following OlMailingAddress constants:
olBusiness, olHome, olNone, or olOther. While it can be changed or entered
independently, any such changes or entries to this property will be overwritten
by any subsequent changes or entries to the property indicated by
SelectedMailingAddress.

Manager Property

Returns an AddressEntry object that represents the manager of the user that
corresponds to this address entry. If the user's manager is not available in the
messaging system, the Manager property returns Nothing .
expression.Manager
expression Required. An expression that returns an AddressEntry object.

ManagerName Property

Returns or s Read/write.	ets a String representing the manager name for the contact.
expression .N	ManagerName
expression	Required. An expression that returns a ContactItem object.

▼ Show All

MarkForDownload Property

Returns or sets an **OlRemoteStatus** constant that determines the status of an item once it is received by a remote user. This property gives remote users with less-than-ideal data-transfer capabilities increased messaging flexibility. Read/write.

OlRemoteStatus can be one of these OlRemoteStatus constants.

olMarkedForCopy The item will be copied to the remote site.

olMarkedForDelete The item will be deleted.

olMarkedForDownload The item will be downloaded in its entirety.

olRemoteStatusNone The item has no remote status.

olUnMarked The item isn't marked for remote status and will be disregarded.

expression. MarkForDownload

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

Example

The following subroutine sets the **MarkForDownload** property of all e-mail items sent by a specified user to **olMarkedForDownload**. All items from the specified user will then be downloaded in their entirety.

MeetingResponseStatus Property

Returns an **OlResponseStatus** constant indicating the overall status of the response to the meeting request for the recipient. Read-only. OlResponseStatus can be one of these OlResponseStatus constants. olResponseAccepted ol Response DeclinedolResponseNone olResponseNotResponded olResponseOrganized olResponseTentative expression.MeetingResponseStatus *expression* Required. An expression that returns a **Recipient** object.

MeetingStatus Property

Returns or sets an <u>OlMeetingStatus</u> constant specifying the meeting status of the appointment. Use this property to make a <u>MeetingItem</u> object available for the appointment. Read/write.

OlMeetingStatus can be one of these OlMeetingStatus constants.

olMeeting olMeetingCanceled olMeetingReceived olNonMeeting

expression. Meeting Status

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.

```
Set myOlApp = CreateObject("Outlook.Application")
Set myItem = myOlApp.CreateItem(olAppointmentItem)
myItem.MeetingStatus = olMeeting
myItem.Subject = "Strategy Meeting"
myItem.Location = "Conference Room B"
myItem.Start = #9/24/1997 1:30:00 PM#
myItem.Duration = 90
Set myRequiredAttendee = myItem.Recipients.Add("Nate Sun")
myRequiredAttendee.Type = olRequired
Set myOptionalAttendee = myItem.Recipients.Add("Kevin Kennedy")
myOptionalAttendee.Type = olOptional
Set myResourceAttendee = myItem.Recipients.Add("Conference Room B")
myResourceAttendee.Type = olResource
myItem.Display
```

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

```
Set myItem = Application.CreateItem(1)
myItem.MeetingStatus = 1
myItem.Subject = "Strategy Meeting"
myItem.Location = "Conference Room B"
myItem.Start = #9/24/97 1:30:00 PM#
myItem.Duration = 90
Set myRequiredAttendee = myItem.Recipients.Add ("Nate Sun")
myRequiredAttendee.Type = 1
Set myOptionalAttendee = myItem.Recipients.Add ("Kevin Kennedy")
myOptionalAttendee.Type = 2
Set myResourceAttendee = myItem.Recipients.Add("Conference Room B")
myResourceAttendee.Type = 3
myItem.Display
```

		_

MemberCount Property

Returns a **Long** indicating the number of members in a distribution list. Readonly.

 $expression. {\bf Member Count}$

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.

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.

		_

Members Property

Returns an AddressEntries collection object (for the AddressEntries object) or a Variant (for the DistListItem object) representing the members (for the AddressEntries object) or a member (for the DistListItem object) of a distribution list in an address book.

A distribution list is an AddressEntry object whose DisplayType property is set to olDistList or olPrivateDistList.

The Members property returns Nothing if the AddressEntry is not a distribution list.

Read-only for the AddressEntries object; read/write for the DistListItem object.

expression.Members

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

▼ Show All

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.

express	ion.I	Messa	igeC	lass
U. IP. UUU			8-2	

jects in the

MiddleName Property

Returns or sets a **String** representing the middle name for the contact. Read/write.

expression. MiddleName

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 .

▼ Show All

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.
purposes of remisuration read write.
expression.Mileage
<i>expression</i> Required. An expression that returns one of the objects in the Applies To list.

MiniIcon Property

MobileTelephoneNumber Property

ing representing the mobile telephone number for the	
expression.MobileTelephoneNumber	
ed. An expression that returns a ContactItem object.	
•	

▼ Show All

ModifiedFormPages Property

Returns the <u>Pages</u> 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 <u>Add</u> method.

expression. ModifiedFormPages

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

Example

This Visual Basic for Applications example returns a new custom page named "My Page" (the same call without a name will return the next available page for customization — for example, "Custom1").

```
Set myOlApp = CreateObject("Outlook.Application")
Set myItem = myOlApp.CreateItem(olContactItem)
Set myPages = myItem.GetInspector.ModifiedFormPages
Set myPage = myPages.Add("My Page")
```

MonthOfYear Property

Returns or sets a Long indicating which month of the year is valid for the specified recurrence pattern. Can be a number from 1 through 12. For example, setting this property to 5 and the RecurrenceType property to olRecursYearly would cause this recurrence pattern to occur every May. Read/write.
expression.MonthOfYear
<i>expression</i> Required. An expression that returns a RecurrencePattern object.
Note This property is only valid for recurrence patterns whose RecurrenceType property is set to olRecursYearly or olRecursYearNth .

▼ Show All

Name Property

Returns or sets the display name for the <u>item</u>. The **Name** property is also the caption for a form. Read/write depending on the object.

expression.Name

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

Note The **Name** property must be set before you can use the **PublishForm** method. It is also necessary for the **Name** property to be set before calling the **Details** method.

Example

This Visual Basic for Applications example uses the <u>CurrentUser</u> property to obtain the name of the currently logged-on user and then displays a message box containing the name.

```
Set myOlApp = CreateObject("Outlook.Application")
Set myNameSpace = myOlApp.GetNameSpace("MAPI")
MsgBox myNameSpace.CurrentUser.Name
```

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

```
Set myNameSpace = Application.GetNameSpace("MAPI")
MsgBox myNameSpace.CurrentUser.Name
```

NetMeetingAlias Property

Returns or sets a Read/write.	String indicating the user's Microsoft NetMeeting ID, or alias.
expression .NetM	[eetingAlias
expression Requ	uired. An expression that returns a ContactItem object.

NetMeetingAutoStart Property

True if this online meeting starts automatically. Read/write Boolean .	
NetMeetingAutoStart	
Required. An expression that returns an AppointmentItem object.	

▼ Show All

NetMeetingDocPathName Property

Returns or sets a String representing the full path to the Microsoft Office document specified for a Microsoft NetMeeting online meeting. Read/write.
expression.NetMeetingDocPathName
<i>expression</i> Required. An expression that returns an AppointmentItem object.

NetMeetingOrganizerAlias Property

Returns or sets a String representing the alias of the meeting organizer, if this is an online meeting. Read/write.
expression.NetMeetingOrganizerAlias
<i>expression</i> Required. An expression that returns an AppointmentItem object.

NetMeetingServer Property

Returns or sets a String specifying the name of the Microsoft NetMeeting server being used for an online meeting. Read/write.
expression.NetMeetingServer
<i>expression</i> Required. An expression that returns an AppointmentItem or ContactItem object.

NetMeetingType Property

Sets or returns an OlNetMeetingType constant specifying the type of Microsoft NetMeeting. Read/write.
OlNetMeetingType can be one of these OlNetMeetingType constants. olChat olNetMeeting olNetShow
expression.NetMeetingType
<i>expression</i> Required. An expression that returns an AppointmentItem object.

NetShowURL Property

NextReminderDate Property

Returns a **Date** that indicates the next time the specified reminder will occur. Read-only.

expression. NextReminderDate

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

Remarks

The **NextReminderDate** property value changes every time the object's **Snooze** method is executed or when the user clicks the **Snooze** button.

Example

End Sub

The following example creates a report of all reminders in the collection and the dates when they will next occur. The subroutine concatenates the **Caption** and **NextReminderDate** properties into a string and displays the string in a dialog box.

```
Sub DisplayNextDateReport()
'Displays the next time all reminders will be displayed
    Dim olApp As Outlook.Application
    Dim objRems As Reminders
    Dim objRem As Reminder
    Dim strTitle As String
    Dim strReport As String
    Set olApp = Outlook.Application
    Set objRems = olApp.Reminders
    strTitle = "Current Reminder Schedule:"
    'Check if any reminders exist
    If objRems.Count = 0 Then
        MsgBox "There are no current reminders."
    Else
        For Each objRem In objRems
            'If string is empty, create new string
            If strReport = "" Then
                strReport = objRem.Caption & vbTab & _
                            objRem. NextReminderDate & vbCr
            Else
                'Add info to string
                strReport = strReport & objRem.Caption & vbTab & _
                            objRem. NextReminderDate & vbCr
            End If
        Next objRem
        'Display report in dialog
        MsgBox strTitle & vbCr & vbCr & strReport
    End If
```

NickName Property

Returns or sets a String representing the nickname for the contact. Read/write.		
expression . I	Nickname	
expression	Required. An expression that returns a ContactItem object.	

▼ Show All

NoAging Property

True to not age the Outlook <u>item</u> . Read/write Boolean .
expression.NoAging
<i>expression</i> Required. An expression that returns one of the objects in the Applies To list.

NoEndDate Property

True if the recurrence pattern has no end date. Read/write **Boolean**.

expression. NoEndDate

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

Remarks

This property must be coordinated with other properties when setting up a
ecurrence pattern. If the PatternEndDate property or the Occurrences
roperty is set, the pattern is considered to be finite and the NoEndDate
roperty is False. If neither PatternEndDate nor Occurrences is set, the pattern
s 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 **RecurrencePattern** object.

Remarks

This property must be coordinated with other properties when setting up a
ecurrence pattern. If the PatternEndDate property or the Occurrences
roperty is set, the pattern is considered to be finite and the NoEndDate
roperty is False. If neither PatternEndDate nor Occurrences is set, the pattern
s 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		
<i>expression</i> Required. An expression that returns a ContactItem object.		

Offline Property

Returns a **Boolean** indicating whether Microsoft Outlook is online (connected to a server) or Offline (not connected to a server). Returns **True** if Outlook is offline. Read-only.

expression. Offline

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

Example

The following function returns **True** or **False** depending on whether the Namespace object is currently online.

```
Sub Off()
'Determines whether Outlook is currently Offline
    Dim olapp As New Outlook.Application
    Dim nmsName As NameSpace
    Set olapp = Outlook.Application
    Set nmsName = olapp.GetNamespace("MAPI")
    Dim blnOff As Boolean
    blnOff = IsOffline(nmsName)
End Sub

Function IsOffline(ByRef nmsName As NameSpace) As Boolean
'Retuns True if Outlook is offline
    IsOffline = nmsName.offline
End Function
```

OneOff Property

	form will be discarded after using once (one-off). False if the form is a custom form. Read/write Boolean .	
expression.OneOff		
expression	Required. An expression that returns a FormDescription object.	

OptionalAttendees Property

Returns or sets a String representing the display string of optional attendees names for the appointment. This property corresponds to the MAPI property PR_DISPLAY_CC. Read/write.		
expression.OptionalAttendees		
expression Required. An expression that returns an AppointmentItem object		

Ordinal Property

Returns or sets a Long specifying the position in the view (ordinal) for the task. Read/write.		
expression . (Ordinal	
expression	Required. An expression that returns a TaskItem object.	

OrganizationalIDNumber Property

Returns or sets a String representing the organizational ID number for the contact. Read/write.		
expression.OrganizationalIDNumber		
<i>expression</i> Required. An expression that returns a ContactItem object.		
expression Required. An expression that returns a ContactItem object.		

Organizer Property

Returns a String representing the name of the organizer of the appointment. Read-only.		
expression .(Organizer	
expression	Required. An expression that returns an AppointmentItem object.	

OriginalDate Property

Returns a **Date** indicating the original date and time of an **AppointmentItem** before it was altered. This property will return the original date even if the **AppointmentItem** has been deleted but it will not return the original time if deletion has occurred. Read-only.

expression.OriginalDate

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

Example

This Visual Basic for Applications example uses <u>CreateItem</u> to create an <u>AppointmentItem</u> object. The <u>RecurrencePattern</u> is obtained for this item using the <u>GetRecurrencePattern</u> method. By setting the <u>RecurrencePattern</u> properties, <u>RecurrenceType</u>, <u>PatternStartDate</u>, and <u>PatternEndDate</u>, the appointments are now a recurring series that occur on a daily basis for the period of one year.

An **Exception** object is created when one instance of this recurring appointment is obtained using the **GetOccurrence** method and properties for this instance are altered. This exception to the series of appointments is obtained using the **GetRecurrencePattern** method to access the **Exceptions** collection associated with this series. Message boxes display the original **Subject** and **OriginalDate** for this exception to the series of appointments and the current date, time, and subject for this exception.

For a description of changes required for this example to work in VBScript, see the Note at the end of the example.

```
Public Sub cmdExample()
    Set myOlApp = New Outlook.Application
    Set myApptItem = myOlApp.CreateItem(olAppointmentItem)
   myApptItem.Start = #2/2/98 3:00:00 PM#
    myApptItem.End = #2/2/98 4:00:00 PM#
   myApptItem.Subject = "Meet with Boss"
    'Get the recurrence pattern for this appointment
    'and set it so that this is a daily appointment
    'that begins on 2/2/98 and ends on 2/2/99
     'and save it.
    Set myRecurrPatt = myApptItem.GetRecurrencePattern
   myRecurrPatt.RecurrenceType = olRecursDaily
   myRecurrPatt.PatternStartDate = #2/2/98#
    myRecurrPatt.PatternEndDate = #2/2/99#
   myApptItem.Save
    'Access the items in the Calendar folder to locate
    'the master AppointmentItem for the new series.
    Set myNamespace = myOlApp.GetNamespace("MAPI")
    Set myFolder = myNamespace.GetDefaultFolder(olFolderCalendar)
```

```
Set myItems = myFolder.Items
    Set myApptItem = myItems("Meet with Boss")
    'Get the recurrence pattern for this appointment
    'and obtain the occurrence for 3/12/98.
    myDate = #3/12/98 \ 3:00:00 \ PM#
    Set myRecurrPatt = myApptItem.GetRecurrencePattern
    Set myOddApptItem = myRecurrPatt.GetOccurrence(myDate)
    'Save the existing subject. Change the subject and
    'starting time for this particular appointment
    'and save it.
    saveSubject = myOddApptItem.Subject
    myOddApptItem.Subject = "Meet NEW Boss"
    newDate = #3/12/98 \ 3:30:00 \ PM#
    myOddApptItem.Start = newDate
    myOddApptItem.Save
    'Get the recurrence pattern for the master
    'AppointmentItem. Access the collection of
    'exceptions to the regular appointments.
    Set myRecurrPatt = myApptItem.GetRecurrencePattern
    Set myException = myRecurrPatt.Exceptions.Item(1)
    'Display the original date, time, and subject
    'for this exception.
    MsgBox myException. Original Date & ": " & saveSubject
    'Display the current date, time, and subject
    'for this exception.
    MsgBox myException.AppointmentItem.Start & ": " & _
myException.AppointmentItem.Subject
End Sub
```

Note For this example to work properly in VBScript, there are only a few simple changes that need to be made in the code.

You don't have to retrieve the application as an object and you must use the values of the constants, so:

```
Set myOlApp = New Outlook.Application
    Set myApptItem = myOlApp.CreateItem(olAppointmentItem)
becomes:
Set myApptItem = Application.CreateItem(1)
```

```
and
myRecurrPatt.RecurrenceType = olRecursDaily
becomes:
myRecurrPatt.RecurrenceType = 0
and
Set myFolder = myNamespace.GetDefaultFolder(olFolderCalendar)
becomes:
Set myFolder = myNamespace.GetDefaultFolder(9)
```

OriginalReminderDate Property

Returns a **Date** that specifies the original date and time that the specified reminder is set to occur. Read-only.

 $expression. {\bf Original Reminder Date}$

expression Required. An expression that returns a **Reminder** 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 example creates a report of all reminders in the **Reminders** collection and the dates at which they are scheduled to occur. The subroutine concatenates the **Caption** and **OriginalReminderDate** properties of all **Reminder** objects in the collection into a string and displays the string in a dialog box.

```
Sub DisplayOriginalDateReport()
'Displays the time at which all reminders will be displayed
    Dim olApp As Outlook.Application
    Dim objRems As Reminders
    Dim objRem As Reminder
    Dim strTitle As String
    Dim strReport As String
    Set olApp = Outlook.Application
    Set objRems = olApp.Reminders
    strTitle = "Original Reminder Schedule:"
    'Check if any reminders exist
    If objRems.Count = 0 Then
        MsgBox "There are no current reminders."
    Else
        For Each objRem In objRems
            'If string is empty, create new string
            If strReport = "" Then
                strReport = objRem.Caption & vbTab & vbTab & _
                            objRem.OriginalReminderDate & vbCr
            Else
                'Add info to string
                strReport = strReport & objRem.Caption & vbTab & vbT
                            objRem. Original Reminder Date & vbCr
            End If
        Next obiRem
        'Display report in dialog
        MsgBox strTitle & vbCr & vbCr & strReport
    End If
End Sub
```

		_

Originator Delivery Report Requested 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 object.	Required. An expression that returns a MailItem or a MeetingItem

OtherAddress Property

Returns or sets a **String** representing the other address for the contact. Read/write.

 $expression. {\bf Other Address}$

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.

 $expression. {\bf Other Address City}$

This property is parsed from the <u>OtherAddress</u> property, but may be changed 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.

 $expression. {\bf Other Address Country}$

This property is parsed from the OtherAddress property, but may be changed or entered independently should it be parsed incorrectly. Note that any such
changes or entries to this property will be overwritten by any subsequent
changes or entries to OtherAddress .

OtherAddressPostalCode Property

Returns or sets a **String** representing the postal code portion of the other address for the contact. Read/write.

 $expression. {\bf Other Address Postal Code}$

This property is parsed from the OtherAddress property, but may be changed or entered independently should it be parsed incorrectly. Note that any such
changes or entries to this property will be overwritten by any subsequent
changes or entries to OtherAddress .

OtherAddressPostOfficeBox Property

Returns or sets a **String** representing the post office box portion of the other address for the contact. Read/write.

 $expression. {\bf Other Address Post Office Box}$

This property is parsed from the OtherAddress property, but may be changed or entered independently should it be parsed incorrectly. Note that any such
changes or entries to this property will be overwritten by any subsequent
changes or entries to OtherAddress .

OtherAddressState Property

Returns or sets a **String** representing the state portion of the other address for the contact. Read/write.

 $expression. {\bf Other Address State}$

This property is parsed from the <u>OtherAddress</u> property, but may be changed 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.

 $expression. {\bf Other Address Street}$

This property is parsed from the <u>OtherAddress</u> property, but may be changed 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 .C	OtherFaxNumber	
expression	Required. An expression that returns a ContactItem object.	

OtherTelephoneNumber Property

Returns or sets a String representing the other telephone number for the contact Read/write.	ct.
expression.OtherTelephoneNumber	
<i>expression</i> Required. An expression that returns a ContactItem object.	

▼ Show All

OutlookInternalVersion Property

Returns a Long representing the build number of the Microsoft Outlook application for an Outlook <u>item</u> . Read-only.
expression.OutlookInternalVersion
<i>expression</i> Required. An expression that returns one of the objects in the Applies To list.

▼ Show All

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

Ownership Property

Returns an OlTaskOwnership specifying the ownership state of the task. Readonly.			
OlTaskOwnership can be one of these OlTaskOwnership constants. olDelegatedTask olNewTask olOwnTask			
expression.Ownership			
expression Required. An expression that returns a TaskItem object.			

PagerNumber Property

Returns or sets a String representing the pager number for the contact. Read/write.						
expression.PagerNumber						
expression	Required. An expression that returns a ContactItem object.					

Panes Property

_

Returns a **Panes** collection object representing the panes displayed by the specified explorer.

expression.Panes

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

Example

This Microsoft Visual Basic/Visual Basic for Applications example adds a group named Marketing as the second group in the Outlook Bar.

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

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

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

Parent Property

Returns the parent **Object** of the specified object. Read-only.

expression.Parent

expression Required. An expression that returns an Outlook object.

Example

This VBScript example uses the **Reply** event and sets the Sent Items folder for the reply item to the folder in which the original item resides.

ction Item_Reply(ByVa Set myResponse.Saves Function	 = Item. Parent	

Password Property

Returns or sets a String specifying the password for modifying the form. Read/write.					
expression.Password					
expression	Required. An expression that returns a FormDescription object.				

PathName Property

Returns a String representing the full path to the linked attached file. This property is only valid for linked files. Read-only.				
expression.PathName				
expression Required. An expression that returns an Attachment object.				

PatternEndDate Property

Returns or sets a **Date** indicating the end date for the recurrence pattern. Read/write.

expression. PatternEndDate

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

Remarks

This property is optional but must be coordinated with other properties when
setting up a recurrence pattern. If this property or the Occurrences property is
set, the pattern is considered to be finite, and the NoEndDate property is False .
If neither PatternEndDate nor Occurrences is set, the pattern is considered
infinite and NoEndDate is True . The Interval property must be set before
setting PatternEndDate .

PatternStartDate Property

Returns or sets a Date indicating the start date for the recurrence pattern. Read/write.	
expression .P	PatternStartDate
expression	Required. An expression that returns a RecurrencePattern object.

PercentComplete Property

Returns or sets a Long indicating the percentage of the task completed at the current date and time. Read/write.	
expression.PercentComplete	
<i>expression</i> Required. An expression that returns a <u>TaskItem</u> object.	

PersonalHomePage Property

Returns or sets a String representing the URL of the personal Web page for the contact. Read/write.	
expression .I	PersonalHomePage
expression	Required. An expression that returns a ContactItem object.

▼ Show All

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

Prefix Property

Returns or sets a String specifying the prefix (for example, "Re") to use with the subject of the item when the action is executed. Note that Outlook automatically adds a colon (:) to the value of the Prefix property when setting the subject of the item. Read/write.		
expression. I	Prefix	
expression	Required. An expression that returns an Action object.	

PrimaryTelephoneNumber Property

Returns or sets a String specifying the primary telephone number for the contact. Read/write.	
expression .P	rimaryTelephoneNumber
expression	Required. An expression that returns a ContactItem object.

ProductCode Property

Returns a String specifying the Microsoft Outlook <u>globally unique identifier</u> (<u>GUID</u>). Read-only.	
expression.ProductCode()	
<i>expression</i> Required. An expression that returns an Application object.	

Profession Property

Returns or sets a String indicating the profession for the contact. Read/write.	
expression . l	Profession
expression	Required. An expression that returns a ContactItem object.

RadioTelephoneNumber Property

Returns or sets a String indicating the radio telephone number for the contact. Read/write.	
expression .I	RadioTelephoneNumber
expression	Required. An expression that returns a ContactItem object.

ReadReceiptRequested Property

True if a read receipt has been requested by the sender. This property corresponds to the MAPI property PR_READ_RECEIPT_REQUESTED. Read/write.	
expression . l	ReadReceiptRequested
expression	Required. An expression that returns a MailItem object.
expression	Required. An expression that returns a Mailitem object.

ReceivedByEntryID Property

Returns a String representing the EntryID for the true recipient as set by the transport provider delivering the mail message. This property corresponds to the MAPI property PR_RECEIVED_BY_ENTRYID. Read-only.	
expression.R	eceivedByEntryID
expression	Required. An expression that returns a MailItem object.

ReceivedByName Property

Returns a String representing the display name of the true recipient for the mail message. This property corresponds to the MAPI property PR_RECEIVED_BY_NAME. Read-only.					
expression.ReceivedByName					
<i>expression</i> Required. An expression that returns a MailItem object.					

ReceivedOnBehalfOfEntryID Property

Returns a **String** representing the **EntryID** of the user delegated to represent the recipient for the mail message. This property corresponds to the MAPI property PR_RCVD_REPRESENTING_ENTRYID. Read-only.

expression.ReceivedOnBehalfOfEntryID

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

ReceivedOnBehalfOfName Property

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
<i>expression</i> 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						
expression Required. An expression that returns a MailItem object.						

▼ Show All

Recipients Property

-

Returns a **Recipients** collection that represents all the recipients for the 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.

Example

This Visual Basic for Applications example creates a new mail message, uses the **Add** method to add "Allison Klein" as a To recipient, and displays the message.

```
Set myOlApp = CreateObject("Outlook.Application")
Set myItem = myOlApp.CreateItem(olMailItem)
Set myRecipient = myItem.Recipients.Add("Allison Klein")
myItem.Display
```

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

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

RecurrenceState Property

Returns an OlRecurrenceState constant indicating the recurrence property of the specified object. Read-only.				
OlRecurrenceState can be one of these OlRecurrenceState constants. olApptException olApptMaster olApptNotRecurring olApptOccurrence				
expression.RecurrenceState				
expression Required. An expression that returns an AppointmentItem object.				
expression Required. An expression that returns an AppointmentItem object.				
expression Required. An expression that returns an AppointmentItem object.				

RecurrenceType Property

Returns or sets an **OlRecurrenceType** constant specifying the frequency of occurrences for the recurrence pattern. Read/write.

OlRecurrenceType can be one of these OlRecurrenceType constants.

olRecursDaily olRecursMonthly olRecursMonthNth olRecursWeekly olRecursYearly olRecursYearNth

expression.RecurrenceType

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

Example

This Visual Basic for Applications example creates a task called "Oil Change" that recurs every three months and uses the **Regenerate** property to set it to regenerate after each recurrence.

```
Set myOlApp = CreateObject("Outlook.Application")
Set myItem = myOlApp.CreateItem(olTaskItem)
Set myPattern = myItem.GetRecurrencePattern
myPattern.RecurrenceType = olRecursMonthly
myPattern.Regenerate = True
myPattern.Interval = 3
myItem.Subject = "Oil Change"
myItem.Save
myItem.Display
```

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

```
Set myItem = Application.CreateItem(3)
Set myPattern = myItem.GetRecurrencePattern
myPattern.RecurrenceType = 2
myPattern.Regenerate = True
myPattern.Interval = 3
myItem.Subject = "Oil Change"
myItem.Save
myItem.Display
```

ReferredBy Property

Returns or sets a String specifying the referral name entry for the contact. Read/write.					
expression.ReferredBy					
expression	Required. An expression that returns a ContactItem object.				

Regenerate Property

True if the task should be regenerated following this pass through the recurrence pattern. This property is used to control the regeneration of the task as each occurrence of a recurring task is completed. Read/write **Boolean**.

expression. Regenerate

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

Example

This Visual Basic for Applications example creates a task called "Oil Change" that recurs every three months and uses the **Regenerate** property to set it to regenerate after each recurrence.

```
Set myOlApp = CreateObject("Outlook.Application")
Set myItem = myOlApp.CreateItem(olTaskItem)
Set myPattern = myItem.GetRecurrencePattern
myPattern.RecurrenceType = olRecursMonthly
myPattern.Regenerate = True
myPattern.Interval = 3
myItem.Subject = "Oil Change"
myItem.Save
myItem.Display
```

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

```
Set myItem = Application.CreateItem(3)
Set myPattern = myItem.GetRecurrencePattern
myPattern.RecurrenceType = 2
myPattern.Regenerate = True
myPattern.Interval = 3
myItem.Subject = "Oil Change"
myItem.Save
myItem.Display
```

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. Reminder Minutes Before Start				
<i>expression</i> Required. An expression that returns an AppointmentItem object.				
1				

ReminderOverrideDefault Property

True if the reminder overrides the default reminder behavior for the appointment, mail item, or task. Read/write Boolean.

Note You must set the ReminderOverrideDefault property to validate the ReminderPlaySound and the ReminderSoundFile properties.

expression.ReminderOverrideDefault

expression Required. An expression that returns an AppointmentItem, MailItem, or TaskItem object.

ReminderPlaySound Property

True if the reminder should play a sound when it occurs for this appointment or task. The ReminderPlaySound property must be set in order to validate the ReminderSoundFile property. Read/write Boolean.

Note This property is only valid if the ReminderOverrideDefault property is set to True.

expression.ReminderPlaySound

expression Required. An expression that returns an AppointmentItem, MailItem, or TaskItem object.

Reminders Property

Returns a **Reminders** collection that represents all current reminders. Read-only.

expression. Reminders

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

Example

The following example returns the **Reminders** collection and displays the captions of all reminders in the collection. If there are no current reminders, a message is displayed to the user.

```
Sub ViewReminderInfo()
'Lists reminder caption information
    Dim olApp As Outlook.Application
    Dim objRem As Reminder
    Dim objRems As Reminders
    Dim strTitle As String
    Dim strReport As String
    Set olApp = Outlook.Application
    Set objRems = olApp.Reminders
    strTitle = "Current Reminders:"
    'If there are reminders, display message
    If olApp.Reminders.Count <> 0 Then
        For Each objRem In objRems
            'If string is empty, create new string
            If strReport = "" Then
                strReport = objRem.Caption & vbCr
            Else
                'Add info to string
                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
```

ReminderSet Property

True if a reminder has been set for this appointment, mail item or task. Read/write **Boolean**.

expression. ReminderSet

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

Example

This example uses the **PropertyChange** event to disable the setting of a reminder on an item.

```
Sub Item_PropertyChange(ByVal myPropertyName)
    Select Case myPropertyName
    Case "ReminderSet"
        MsgBox "You may not set a reminder on this item!"
        Item.ReminderSet = False
        Case Else
    End Select
End Sub
```

ReminderSoundFile Property

Returns or sets a **String** indicating the path and file name of the sound file to play when the reminder occurs for the appointment, mail message, or task. This property is only valid if the **ReminderOverrideDefault** and **ReminderPlaySound** properties are set to **True**. Read/write.

expression.ReminderSoundFile

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

▼ Show All

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 MailItem, MeetingItem or TaskItem object.

RemoteMessageClass Property

Returns a S t	tring indicating the message class for the remote item. Read-only.
expression. I	RemoteMessageClass
expression	Required. An expression that returns a RemoteItem object.

RemoteStatus Property

Returns or sets an OlRemoteStatus constant specifying the remote status of the mail message. Read/write.
OlRemoteStatus can be one of these OlRemoteStatus constants. olMarkedForCopy olMarkedForDelete olMarkedForDownload olRemoteStatusNone olUnMarked
expression.RemoteStatus
<i>expression</i> Required. An expression that returns a MailItem object.

ReplyRecipientNames Property

message. Th	micolon-delimited String list of reply recipients for the mail his property only contains the display names for the reply recipients. cipients list should be set by using the ReplyRecipients collection.
expression. F	ReplyRecipientNames
expression	Required. An expression that returns a MailItem object.

ReplyRecipients Property

Returns a R the mail mes	ecipients collection that represents all the reply recipient objects for ssage.
expression .I	ReplyRecipients
expression	Required. An expression that returns a MailItem object.

ReplyStyle Property

Returns or sets an **OlActionReplyStyle** constant indicating the text formatting reply style for the specified action. Read/write. OlActionReplyStyle can be one of these OlActionReplyStyle constants. olEmbedOriginalItem olIncludeOriginalText olIndentOriginalText olLinkOriginalItem olOmitOriginalText olReplyTickOriginalText olUserPreference expression.ReplyStyle *expression* Required. An expression that returns an **Action** object.

ReplyTime Property

Returns or sets a Date indicating the reply time for the appointment. Read/write.		
expression. I	ReplyTime	
expression	Required. An expression that returns an AppointmentItem object.	

RequiredAttendees Property

meeting app	emicolon-delimited String of required attendee names for the cointment. This property only contains the display names for the endees. The attendee list should be set by using the Recipients Read/write.
expression. I	RequiredAttendees
expression	Required. An expression that returns an AppointmentItem object.

Resolved Property

True if the recipient has been validated against the Address Book. Read-only **Boolean**.

expression. Resolved

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

This Visual Basic for Applications example uses the <u>GetSharedDefaultFolder</u> method to resolve the <u>Recipient</u> object representing Kim Buhler, and then returns her shared default Calendar folder.

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.

Resources Property

property contains the display names only. The **Recipients** collection should be used to modify the resource recipients. Resources are added as **BCC** recipients to the collection. Read/write.

expression.Resources

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

Returns a semicolon-delimited **String** of resource names for the meeting. This

ResponseRequested Property

True if the sender would like a response to the meeting request for the appointment. Read/write Boolean .		
I		
ression that returns an AppointmentItem object.		

ResponseState Property

Returns or sets 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
<i>expression</i> Required. An expression that returns a TaskItem object.

ResponseStatus Property

Returns an **OlResponseStatus** constant indicating the overall status of the meeting for the current user for the appointment. Read-only. OlResponseStatus can be one of these OlResponseStatus constants. olResponseAccepted olResponseDeclined olResponseNone olResponseNotResponded olResponseOrganized olResponseTentative expression.ResponseStatus Required. An expression that returns an **AppointmentItem** object. expression

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

Results Property

Returns a **Results** collection that specifies the results of the search.

expression. Results

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

The following event procedure example stores the results of a search in a variable called objRsts and prints the results of the search in the Immediate window. Always use the AdvancedSearchComplete event to ensure the results of the search.

Private Sub Application_AdvancedSearchComplete(ByVal SearchObject As

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

End Sub

Role Property

Returns or sets a String containing the free-form text string associating the owner of a task with a role for the task. Read/write.	
expression.Role	
expression Required. An expression that returns a TaskItem object.	

Saved Property

True if the Microsoft Outlook item has not been modified since the last save. Read-only **Boolean**.

expression.Saved

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

This example tests for the <u>Close</u> event and if the item has not been saved, it uses the <u>Save</u> method to save the item without prompting the user.

]	tion Item_Close() If not Item.Saved Item.Save End If	Then
	-unction	

▼ Show All

SaveOption Property

Returns an <u>OlViewSaveOption</u> 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.

End Sub

The following example displays the names of all views that can be accessed by all users in the Notes folder.

```
Sub DisplayPublicViews()
'Displays the names of all views that are available to
'all users of this folder
    Dim olApp As Outlook.Application
    Dim objName As NameSpace
    Dim objViews As Views
    Dim objView As View
    Dim strSaveOption As String
    Set olApp = Outlook.Application
    Set objName = olApp.GetNamespace("MAPI")
    Set objViews = objName.GetDefaultFolder(olFolderNotes).Views
    For Each objView In objViews
        If objView. SaveOption = olViewSaveOptionThisFolderEveryone T
            'if list is empty, start list
            If strSaveOption = "" Then
                strSaveOption = "The following views are available"
                                " to all users: " & vbCr & vbCr & obj
            Else
            'If not empty, add names to list
                strSaveOption = strSaveOption &
                                objView.Name & vbCr
            End If
        End If
    Next objView
    'If views exist display names
    If strSaveOption <> "" Then
        MsgBox strSaveOption
    End If
```

SaveSentMessageFolder Property

Returns a **MAPIFolder** object that represents the folder in which a copy of the mail message will be saved upon being sent.

expression. Save Sent Message Folder

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

This VBScript example uses the **Reply** event and sets the Sent Items folder for the reply item to the folder in which the original item resides.

ction Item_Reply Set myResponse. Function	 •	Item.Parent	

SchedulePlusPriority Property

Returns or sets a String representing the Microsoft Schedule+ priority for the task. Can be 1 through 9, A through Z, or A1 through Z9. Priority 1 is the highest. Read/write.			
expression.SchedulePlusPriority			
expression Required. An expression that returns a TaskItem object.			

Scope Property

Returns a **String** that specifies the scope of the specified search. Read-only.

expression.Scope

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

Remarks

The scope of the search is defined when the search is initiated. For more information, see the <u>AdvancedSearch</u> method.

The following subroutine creates a **Search** object. The user's Inbox is specified as the scope of the search. The event subroutine occurs when the search has completed and displays the <u>Tag</u> and **Scope** properties for the new object as well as the results of the search.

```
Sub SearchInboxFolder()
'Searches the Inbox

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

The **AdvancedSearchComplete** event is used to capture the results of the search.

Private Sub Application_AdvancedSearchComplete(ByVal SearchObject As

End Sub

ScriptText Property

Returns a **String** containing all the VBScript in the form's Script Editor. Readonly.

expression.ScriptText

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

This VBScript example uses the **Open** event to access the **HTMLBody** property of an item. This sets the **EditorType** property of the item's **Inspector** to **olEditorHTML**. When the item's **Body** property is set, the **EditorType** property is changed to the default. For example, if the default editor is set to RTF, the **EditorType** is set to **olEditorRTF**.

If this code is placed in the Script Editor of a form in design mode, the message boxes during run time will reflect the change in the **EditorType** as the body of the form changes. The final message box utilizes the **ScriptText** property to display all the VBScript code in the Script Editor.

```
Function Item_Open()
    'Set the HTMLBody of the item.
    Item.HTMLBody = "<HTML><H2>My HTML page.</H2><BODY>My body.</BOD</pre>
    '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 **Search** object.

Remarks

If **True**, the **S earch** object searched through any subfolders in the specified filter path.

The following example creates a **Search** object that contains all items in the user's Inbox with the subject Office Christmas Party. The **SearchSubFolders** argument is specified as **True**.

End Sub

Use an **AdvancedSearchComplete** event procedure to display the results of the search.

```
Private Sub Application_AdvancedSearchComplete(ByVal SearchObject As Dim objRsts As Results

MsgBox "The search " & SearchObject.Tag & "has completed. The Se "was set to " & SearchObject.SearchSubFolders & "."

Set objRsts = SearchObject.Results
'Print out number in Results collection
Debug.Print objRsts.Count
'Print out each member of Results collection
For Each Item In objRsts
Debug.Print Item
Next
```

End Sub

		_

SelectedMailingAddress Property

Returns or sets an OlMailingAddress constant indicating the type of the mailing address for the contact. Read/write.
OlMailingAddress can be one of these OlMailingAddress constants. olBusiness olHome olNone olOther
expression.SelectedMailingAddress
<i>expression</i> Required. An expression that returns a ContactItem object.

Selection Property

Returns a **Selection** object consisting of one or more items selected in the current view.

expression. Selection

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

Remarks

If the current folder is a file-system folder, or if Outlook Today or any folder with a current Web view is currently displayed, this property returns an empty collection.

The following Microsoft Visual Basic/Visual Basic for Applications example uses the **Count** property and **Item** method of the **Selection** collection returned by the **Selection** property to display the senders of all messages selected in the active explorer window.

```
Dim myOlApp As New Outlook.Application
Dim myOlExp As Outlook.Explorer
Dim myOlSel As Outlook.Selection
Dim MsgTxt As String
MsgTxt = "You have selected items from" & Chr(13) & Chr(13)
Set myOlExp = myOlApp.ActiveExplorer
Set myOlSel = myOlExp.Selection
For x = 1 To myOlSel.Count
    MsgTxt = MsgTxt & myOlSel.Item(x).SenderName & Chr(13)
Next x
MsgBox MsgTxt
```

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

SenderName Property

Returns a String indicating the display name of the sender for the mail message, meeting item or post. This property corresponds to the MAPI property PR_SENDER_NAME. Read-only.
expression.SenderName
<i>expression</i> Required. An expression that returns a <u>MailItem</u> , <u>MeetingItem</u> or <u>PostItem</u> object.

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 *MeetingItem** or *MailItem** object.

SentOn Property

Returns a **Date** indicating the date and time on which the mail message, meeting item, or post was sent. This property corresponds to the MAPI property PR_CLIENT_SUBMIT_TIME. When you send a meeting request item using the object's **Send** method, the transport provider sets the **ReceivedTime** and **SentOn** properties for you. Read-only.

expression.SentOn

expression Required. An expression that returns a **MailItem**, **MeetingItem** or **PostItem** object.

SentOnBehalfOfName Property

Returns a String indicating the display name for the intended sender of the mail message. This property corresponds to the MAPI property PR_SENT_REPRESENTING_NAME. Read/write.		
expression.SentOnBehalfOfName		
<i>expression</i> Required. An expression that returns a MailItem object.		

▼ Show All

Session Property

-		
Returns the NameSpace object for the current <u>session</u> .		
expression.Session		
<i>expression</i> Required. An expression that returns one of the objects in the Applies To list to which the Session property is being applied.		

▼ Show All

Shortcuts Property

_

Returns an **OutlookBarShortcuts** collection of **shortcuts** contained within the specified **Outlook Bar** group.

expression.Shortcuts

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

This Microsoft Visual Basic/Visual Basic for Applications example deletes all empty groups in the Outlook Bar.

```
Dim myOlApp As New Outlook.Application
Dim myOlBar As Outlook.OutlookBarPane
Dim myOlGroup As Outlook.OutlookBarGroup
Set myOlBar = myOlApp.ActiveExplorer.Panes.Item("OutlookBar")
For x = myOlBar.Contents.Groups.Count To 1 Step -1
    Set myOlGroup = myOlBar.Contents.Groups.Item(x)
    If myOlGroup.Shortcuts.Count = 0 Then
        myOlBar.Contents.Groups.Remove x
    End If
Next x
```

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

```
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
```

ShowAsOutlookAB Property

Returns or sets a read-only **Boolean** variable that specifies whether the folder will be displayed as an Outlook Address Book. Read-write.

expression. ShowAsOutlookAB

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

The following example creates a reference to a new folder and modifies its **ShowAsOutlookAB** property so that it is displayed as an Address Book.

```
Sub FaveChange()

Dim olApp As Outlook.Application
Dim nmsName As namespace
Dim fldFolder As MAPIFolder

Set olApp = Outlook.Application
'Create instance of namespace
Set nmsName = olApp.GetNamespace("Mapi")
Set fldFolder = nmsName.GetDefaultFolder(olFolderInbox)

'Display the folder as Outlook Address Book
fldFolder.ShowAsOutlookAB = True

End Sub
```

ShowOn Property

Returns or sets 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		
<i>expression</i> Required. An expression that returns an Action object.		

Size Property

Returns a Long indicating the size (in bytes) of the Outlook item. Read-only.
expression .Size
expression Required. An expression that returns one of the objects in the Applies To list.

Spouse Property

Returns or sets a String indicating the spouse name entry for the contact. Read/write.		
expression.S	pouse	
expression	Required. An expression that returns a ContactItem object.	

Standard Property

Returns a Boolean that specifies whether or not the view is a built-in Microsoft Outlook view.		
expression. Standard		
expression Required. An expression that returns a <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 **AppointmentItem** or **JournalItem** 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.

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

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

```
Set myItem = Application.CreateItem(1)
myItem.MeetingStatus = 1
myItem.Subject = "Strategy Meeting"
myItem.Location = "Conference Room B"
myItem.Start = #9/24/97 1:30:00 PM#
myItem.Duration = 90
Set myRequiredAttendee = myItem.Recipients.Add ("Nate Sun")
myRequiredAttendee.Type = 1
Set myOptionalAttendee = myItem.Recipients.Add ("Kevin Kennedy")
myOptionalAttendee.Type = 2
Set myResourceAttendee = myItem.Recipients.Add("Conference Room B")
myResourceAttendee.Type = 3
myItem.Display
```

StartDate Property

Returns or sets a Date specifying the starting date and time for the specified task. Read/write.		
expression .S	tartDate	
expression	Required. An expression that returns a TaskItem object.	

StartTime Property

Returns or sets a Date indicating the start time for a given occurrence of the recurrence pattern. This property is only valid for appointments. Read/write.	
expression.StartTime	
<i>expression</i> Required. An expression that returns a RecurrencePattern object	t.

Status Property

Returns or sets an <u>OlTaskStatus</u> constant specifying the status for the task. Corresponds to the Status field of a <u>TaskItem</u> . Read/write.
OlTaskStatus can be one of these OlTaskStatus constants. olTaskComplete olTaskDeferred olTaskInProgress olTaskNotStarted olTaskWaiting
expression. Status expression Required. An expression that returns a TaskItem object.

StatusOnCompletionRecipients Property

Returns or sets a semicolon-delimited **String** of display names for recipients who will receive status upon completion of the task. This property is calculated from the **Recipients** property. Recipients returned by the

StatusOnCompletionRecipients property correspond to BCC recipients in the **Recipients** collection. Read/write.

expression.StatusOnCompletionRecipients

expression	Required. An expression that returns a TaskItem object.

StatusUpdateRecipients Property

Returns a semicolon-delimited String of display names for recipients who receive status updates for the task. This property is calculated from the Recipients property. Recipients returned by the StatusUpdateRecipients property correspond to CC recipients in the Recipients collection. Read-only.	•
expression.StatusUpdateRecipients	
<i>expression</i> Required. An expression that returns a TaskItem object.	

StoreID Property

Returns a **String** indicating the store ID for the folder. Read-only.

expression. StoreID

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

This Visual Basic for Applications example obtains the **EntryID** and **StoreID** for a subfolder within the Tasks **Folders** collection and then calls the **GetFolderFromID** method using these values to obtain the same subfolder. The folder is then displayed.

```
Set myOlApp = CreateObject("Outlook.Application")
Set myNameSpace = myOlApp.GetNameSpace("MAPI")
Set myFolder = myNameSpace.GetDefaultFolder(olFolderTasks)
Set myOwnFolder = myFolder.Folders(1)
myEntryID = myOwnFolder.EntryID
myStoreID = myOwnFolder.StoreID
Set myNewFolder = myNameSpace.GetFolderFromID(myEntryID, myStoreID)
myNewFolder.Display
```

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

```
Set myNameSpace = Application.GetNameSpace("MAPI")
Set myFolder = myNameSpace.GetDefaultFolder(13)
Set myOwnFolder = myFolder.Folders(1)
myEntryID = myOwnFolder.EntryID
myStoreID = myOwnFolder.StoreID
Set myNewFolder = myNameSpace.GetFolderFromID(myEntryID, myStoreID)
myNewFolder.Display
```

Subject Property

Returns or sets a **String** indicating the subject for the Microsoft Outlook item. This property corresponds to the MAPI property PR_SUBJECT. The **Subject** property is the default property for Outlook items. Read/write.

Note For a **NoteItem** object, the **Subject** property is a read-only **String** that is calculated from the body text of the note.

expression.Subject

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

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

```
Set myOlApp = CreateObject("Outlook.Application")
Set myNamespace = myOlApp.GetNamespace("MAPI")
Set myFolder = myNamespace.GetDefaultFolder(olFolderInbox)
Set myNewFolder = myFolder.Folders.Add("Saved Mail", olFolderDrafts)
Set myItem = myOlApp.CreateItem(olMailItem)
myItem.Subject = "Speeches"
Set myCopiedItem = myItem.Copy
myCopiedItem.Move myNewFolder
```

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

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

▼ Show All

Submitted Property

Returns a Boolean value that indicates if the item has been <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	
<i>expression</i> 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
<i>expression</i> Required. An expression that returns a ContactItem object.
Note The <u>LastName</u> , <u>FirstName</u> , <u>MiddleName</u> , and <u>Suffix</u> properties are parsed from the <u>FullName</u> property.

SyncObjects Property

Returns a **SyncObjects** collection containing all synchronization profiles.

expression.SyncObjects

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

Example

This Microsoft Visual Basic/Visual Basic for Applications example starts synchronization of the user's second synchronization profile in response to the user clicking a button on a form.

```
Private Sub Command1_Click()
    Dim myOlApp As New Outlook.Application
    Dim mySyncs As Outlook.SyncObjects
    Set mySyncs = myOlApp.GetNamespace("MAPI").SyncObjects
    Set mySyncObj = mySyncs.Item(1)
    mySyncObj.Start
End Sub
```

Tag Property

_

Returns a **String** specifying the name of the current search. The **Tag** property is used to identify a specific search.

expression.Tag

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

Remarks

The **Tag** property is set by the **AdvancedSearch** method when the **Search** object is created.

Example

The following example searches through the user's inbox for all items that do not have a flag. The name "FlagSearch", specified by the Tag property, is given to the search.

The following example uses the **AdvancedSearchComplete** event to capture the outcome of the search. The **Tag** property of the search is displayed to notify the user which search has completed.

```
Private Sub Application_AdvancedSearchComplete(ByVal SearchObject As 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
```

▼ Show All

Target Property

_

Returns a **Variant** indicating the target of the specified shortcut in an <u>Outlook</u> <u>Bar</u> group. Read-only.

expression.Target

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

Remarks

The return type depends on the shortcut type. If the shortcut represents a Microsoft Outlook folder, the return type is **MAPIFolder**. If the shortcut represents a file-system folder, the return type is an **Object**. If the shortcut represents a file-system path or URL, the return type is a **String**.

Example

This Microsoft Visual Basic/Visual Basic for Applications example steps through the shortcuts in the first Outlook Bar group. If it finds a shortcut that is not an Outlook folder, it deletes it.

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

TeamTask Property

True if the task is a team task. Read/write Boolean .		
expression.TeamTask		
expression	Required. An expression that returns a <u>TaskItem</u> object.	

TelexNumber Property

Returns or sets a String indicating the telex number for the contact. Read/write.	
expression. 1	TelexNumber
expression	Required. An expression that returns a ContactItem object.

Template Property

Returns or se Read/write.	ets a String indicating the name of the template for the form.		
1	Note This property refers to a Microsoft Word .DOT template file, and so is applicable for forms with UseWordMail = True .		
expression. T	emplate		
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 Title
expression	Required. An expression that returns a ContactItem object.

To Property

Returns or sets a semicolon-delimited String list of display names for the To recipients for the Outlook item. This property contains the display names only. The To property corresponds to the MAPI property PR_DISPLAY_TO. The Recipients collection should be used to modify this property. Read/write.		
expression. T	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	
<i>expression</i> Required. An expression that returns an Explorer , Inspector , or NoteItem object.	

TotalWork Property

Returns or sets a Long indicating the total work for the task. Corresponds to the Total work field on the Details tab of a Task item. Read/write.	
expression. TotalWork	
expression Required. An expression that returns a TaskItem object.	

TrackingStatus Property

Returns or sets an **OlTrackingStatus** constant indicating the tracking status for the recipient. Read/write. OlTrackingStatus can be one of these OlTrackingStatus constants. olTrackingDelivered olTrackingNone olTrackingNotDelivered olTrackingNotRead olTrackingRead olTrackingRecallFailure olTrackingRecallSuccess olTrackingReplied expression.TrackingStatus *expression* Required. An expression that returns a **Recipient** object.

TrackingStatusTime Property

Returns or sets a Date indicating the tracking status date and time for the recipient. Read/write.	
expression.TrackingStatusTime	
<i>expression</i> Required. An expression that returns a Recipient object.	

TransferSize Property

Returns a Long specifying the transfer size (in bytes) for the remote item. Readonly.
expression.TransferSize
<i>expression</i> Required. An expression that returns a RemoteItem object.

TransferTime Property

Returns a Long indicating the transfer time (in seconds) for the remote i Read-only.	tem.
expression.TransferTime	
<i>expression</i> Required. An expression that returns a RemoteItem object.	

TTYTDDTelephoneNumber Property

▼ Show All

Type Property

_

▶ Type property as it applies to the **Attachment** object.

Returns an **OlAttachmentType** constant indicating the type of the specified object. Read-only.

OlAttachmentType can be one of these OlAttachmentType constants.

olByReference olByValue olEmbeddeditem olOLE

expression. Type

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

▶ Type property as it applies to the **Link** object.

Returns an <u>OlObjectClass</u> constant indicating the type of item represented by the **Link** object. Read-only.

OlObjectClass can be one of these OlObjectClass constants.

olAction

olActions

olAddressEntries

olAddressEntry

olAddressList

olAddressLists

olApplication

olAppointment

olAttachment

olAttachments

olContact

olDistributionList

olDocument

olException

olExceptions

olExplorer

olExplorers

olFolder

olFolders

olFormDescription

olInspector

olInspectors

olItemProperties

olItemProperty

olItems

olJournal

olLink

olLinks

olMail

olMeeting Cancellation

olMeetingRequest

ol Meeting Response Negative

ol Meeting Response Positive

olMeetingResponseTentative

olNamespace

olNote

olObjects

olOutlookBarGroup

ol Out look Bar Groups

olOutlookBarPane

ol Outlook Bar Short cut

```
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
```

 $expression. {\bf Type}$

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

ol Keywords

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 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.

```
Set myOlApp = CreateObject("Outlook.Application")
Set myItem = myOlApp.CreateItem(olAppointmentItem)
myItem.MeetingStatus = olMeeting
myItem.Subject = "Strategy Meeting"
myItem.Location = "Conference Room B"
myItem.Start = #9/24/97 1:30:00 PM#
myItem.Duration = 90
Set myRequiredAttendee = myItem.Recipients.Add ("Nate Sun")
myRequiredAttendee.Type = olRequired
```

```
Set myOptionalAttendee = myItem.Recipients.Add ("Kevin Kennedy")
myOptionalAttendee.Type = olOptional
Set myResourceAttendee = myItem.Recipients.Add("Conference Room B")
myResourceAttendee.Type = olResource
myItem.Send
```

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

```
Set myItem = Application.CreateItem(1)
myItem.MeetingStatus = 1
myItem.Subject = "Strategy Meeting"
myItem.Location = "Conference Room B"
myItem.Start = #9/24/97 1:30:00 PM#
myItem.Duration = 90
Set myRequiredAttendee = myItem.Recipients.Add ("Nate Sun")
myRequiredAttendee.Type = 1
Set myOptionalAttendee = myItem.Recipients.Add ("Kevin Kennedy")
myOptionalAttendee.Type = 2
Set myResourceAttendee = myItem.Recipients.Add("Conference Room B")
myResourceAttendee.Type = 3
myItem.Send
```

UnRead Property

True if the Microsoft Outlook item has not been opened (read). Read/write Boolean .
expression.UnRead
<i>expression</i> Required. An expression that returns one of the objects in Applies Tollist.

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

User2 Property

Returns or sets a String specifying the second Microsoft Schedule+ user for the contact. Read/write.
expression.User2
<i>expression</i> Required. An expression that returns a ContactItem object.

User3 Property

User4 Property

Returns or sets a String specifying the fourth Microsoft Schedule+ user for the contact. Read/write.
expression.User4
<i>expression</i> Required. An expression that returns a ContactItem object.

UserCertificate Property

Returns or sets a String containing the user's authentication certificate for the contact. Read/write.					
expression.UserCertificate					
<i>expression</i> Required. An expression that returns a ContactItem object.					

UserProperties Property

Returns the <u>UserProperties</u> collection that represents all the user properties for the Outlook item.

expression. User Properties

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

This VBScript example uses the **CustomPropertyChange** event to enable a control when a **Boolean field** is set to **True**.

For this example, create two custom fields on the second page of a form. The first, a **Boolean** field is named "RespondBy". The second field is named "DateToRespond".

```
Sub Item_CustomPropertyChange(ByVal myPropName)
    Select Case myPropName
        Case "RespondBy"
            Set myPages = Item.GetInspector.ModifiedFormPages
            Set myCtrl = myPages("P.2").Controls("DateToRespond")
            If Item.UserProperties("RespondBy").Value Then
                myCtrl.Enabled = True
                myCtrl.Backcolor = 65535
                                                'Yellow
            Else
                myCtrl.Enabled = False
                myCtrl.Backcolor = 0 'Black
            End If
        Case Else
    End Select
End Sub
```

UseWordMail Property

True to use Microsoft Word as the default editor for the form. Read/write Boolean .					
expression.UseWordMail					
<i>expression</i> Required. An expression that returns a FormDescription object.					

ValidationFormula Property

ValidationText Property

Returns or sets a String specifying the validation text for the specified user property. Read/write.					
expression.ValidationText					
<i>expression</i> Required. An expression that returns a UserProperty object.					

Value Property

Returns or sets a Variant indicating the value for the specified user property. Read/write.
expression.Value
<i>expression</i> Required. An expression that returns one of the objects in the Applies To list.

Version Property

Returns or sets a String indicating the number of the version. Read/write for the FormDescription object; read-only for all other objects in the Applies To list.					
expression.Version					
<i>expression</i> Required. An expression that returns one of the objects in the Applies To list.					

Views Property

_

Returns a <u>Views</u> collection. Each **View** object in the **Views** collection defines the manner in which items are displayed in an Explorer.

expression.Views

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

The following example creates an instance of the **Views** collection and returns a view called "Table View".

```
Sub GetView()
'Returns a view object

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

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

End Sub
```

▼ Show All

ViewType Property

-

▶ <u>ViewType property as it applies to the **OutlookBarGroup** object.</u>

Returns or sets the icon view displayed by the specified Outlook bar group. Read/write OlOutlookBarViewType.

OlOutlookBarViewType can be one of these OlOutlookBarViewType constants.

olLargeIcon olSmallIcon

expression.ViewType

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

▶ <u>ViewType property as it applies to the **View** object.</u>

Returns an **OlViewType** constant that represents the type of the current view. Read-only.

OlViewType can be one of these OlViewType constants.

olCalendarView olCardView olIconView olTableView olTimelineView

expression.ViewType

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

▶ As it applies to the **OutlookBarGroup** object.

The following Microsoft Visual Basic/Visual Basic for Applications example toggles the first Outlook bar group between large and small icon views.

```
Dim myOlApp As New Outlook.Application
Dim myOlBar As Outlook.OutlookBarPane
Dim myOlGroup As Outlook.OutlookBarGroup
Set myOlBar = myOlApp.ActiveExplorer.Panes.Item("OutlookBar")
Set myOlGroup = myOlBar.Contents.Groups.Item(1)
If myOlGroup.ViewType = olLargeIcon Then
    myOlGroup.ViewType = olSmallIcon
Else
    myOlGroup.ViewType = olLargeIcon
End If
```

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

```
Set myOlBar = Application.ActiveExplorer.Panes.Item("OutlookBar")
Set myOlGroup = myOlBar.Contents.Groups.Item(1)
If myOlGroup.ViewType = 0 Then
    myOlGroup.ViewType = 1
Else
    myOlGroup.ViewType = 0
End If
```

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

The following example displays the name and type of all views in the user's inbox.

```
Sub DisplayViewMode()
'Displays the names and view modes for all views

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

```
Dim strTypes As String

Set olApp = Outlook.Application
Set objName = olApp.GetNamespace("MAPI")
Set objViews = objName.GetDefaultFolder(olFolderInbox).Views

'Collect names and view types for all views
For Each objView In objViews
    strTypes = strTypes & objView.Name & vbTab & vbTab & objView
Next objView

'Display message box
MsgBox "Current Inbox Views and Viewtypes:" & 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 example toggles the visible state of the Outlook Bar.

```
Dim myOlApp As New Outlook.Application
Dim myOlBar As Outlook.OutlookBarPane
Set myOlBar = myOlApp.ActiveExplorer.Panes.Item("OutlookBar")
myOlBar.Visible = Not myOlBar.Visible
```

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

```
Set myOlBar = Application.ActiveExplorer.Panes.Item("OutlookBar")
myOlBar.Visible = Not myOlBar.Visible
```

VotingOptions Property

Returns or sets a String specifying a delimited string containing the voting options for the mail message. Read/write.					
expression.VotingOptions					
at returns a <u>MailItem</u> object.					

VotingResponse Property

Returns or sets a String specifying the voting response for the mail message. This property is usually set to one of the delimited values returned by the VotingOptions property on a reply to the original message. Read/write.					
expression.VotingResponse					
<i>expression</i> Required. An expression that returns a MailItem object.					

▼ Show All

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 ContactItem object.					

▼ Show All

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. Web View On

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

Remarks

This property empty.	is always Fa l	l se if the valu	ie of the Wel	bViewURL p	property is

▼ Show All

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.		

Width Property

Returns or sets a Long indicating the width (in pixels) of the specified object. Read/write.		
Width		
Required. An expression that returns one of the objects in the Applies		

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 **Count** property and **Item** method of the **Explorers** collection to enumerate the open explorer windows.

```
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
```

For x = 1 To Application. Explorers. Count

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.

```
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.

The following VBScript example demonstrates the use of Word's object model. It adds a filled rectangle to the message area of the item.

```
Sub CommandButton1_Click()
    Set myInspector = Item.GetInspector
    Set WordDoc = myInspector.WordEditor
    Set Fill = WordDoc.Shapes.AddShape(1, 90, 90, 90, 50).Fill 'ms
    Fill.ForeColor.RGB = RGB(128, 0, 0)
    Fill.BackColor.RGB = RGB(170, 170, 170)
    Fill.TwoColorGradient 4, 1 'msoGradientDiagonalDown=4
End Sub
```

XML Property

Returns or sets a value that specifies the XML definition of the current view. The XML definition describes the view type by using a series of tags and keywords corresponding to various properties of the view itself. When the view is created, the XML definition is parsed to render the settings for the new view. Read/write **String**.

expression.XML

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

End Sub

The following example creates a new view and displays its XML definition in the Immediate Window.

```
Sub XMLView()
'Creates a new view and displays the XML definition
'in the immediate window

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

Set olApp = Outlook.Application
Set objViews = _
olApp.GetNamespace("mapi").GetDefaultFolder(olFolderInbox).Views
Set objView = objViews.Add _
("New Table View", olTableView, olViewSaveOptionAllFoldersOfT

Debug.Print objView.XML
```

The modified properties which are visible in the following XML source code. In addition to the property definitions, the XML source also defines any objects that make up the view. The following example displays the XML definition of columns that appear in the above view.

 <style>padding-left:3px;text-align:center;padding-left:3px</style> 	

YomiCompanyName Property

Returns or sets a String indicating the Japanese phonetic rendering (yomigana) of the company name for the contact. Read/write.				
objContactItem.YomiCompanyName				
objContactItem 	Required. An expression that returns a ContactItem object.			

YomiFirstName Property

Returns or sets a String indicating the Japanese phonetic rendering (yomigana) of the first name for the contact. Read/write.				
objContactItem.YomiFirstName				
objContactItem	Required. An expression that returns a ContactItem object.			

YomiLastName Property

Returns or sets a String indicating the Japanese phonetic rendering (yomigana) of the last name for the contact. Read/write.				
objContactItem.YomiLastName				
objContactItem Required. An expression that returns a ContactItem object.				

Activate Event

-

Occurs when an explorer or inspector becomes the active window, either as a result of user action or through program code. This event is not available in VBScript.

Sub object_Activate()

object An expression that evaluates to an **Explorer** or **Inspector** object.

This code example uses the <u>WindowState</u> property to maximize the topmost explorer window when the <u>Activate</u> 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 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 **Application** 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 was executed programmatically.

The following example displays the number of objects returned by the search.

```
Private Sub olApp_AdvancedSearchComplete(ByVal SearchObject As Searc 'Get Search object's results collection.

Dim myResults As Results

Set myResults = SearchObject.Results

'Display the number of objects found in the search

MsgBox "The number of objects found in the search is: " & my
End Sub
```

AdvancedSearchStopped Event

Occurs when a specified **Search** object's **Stop** method has been executed.

Private Sub application_ AdvancedSearchStopped(ByVal SearchObject As Object)

expression A variable that references an object of type **Application** 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 **Results** collection will no longer be updated. This event can only be triggered programmatically.

The follo	owing example di	splays the na	me of the So	earch obje	ect that was st	opped.
Private	Sub olApp_Adva 'Inform the us MsgBox "An Adv "The resul	er that the ancedSearch	e search ha n has been	as stoppe stopped.	ed.	
End Sub					-	

AttachmentAdd Event

Occurs when an attachment has been added to an item.

Sub object_AttachmentAdd(Attachment As Attachment)

object An object that evaluates to one of the objects in the Applies To list. In VBScript, use the word **Item**.

Attachment Required. The **Attachment** that was added to the item.

This VBScript example checks the size of the item after an embedded attachment has been added and displays a warning if the size exceeds 500,000 bytes.

AttachmentRead Event

Occurs when an attachment in an item has been opened for reading.

Sub object_AttachmentRead(ByVal Attachment As Attachment)

object An object that evaluates to one of the objects in the Applies To list. In VBScript, use the word **Item**.

Attachment Required. The **Attachment** that was opened.

This VBScript example reminds the user to also save changes to the original of an embedded file.

```
Sub Item_AttachmentRead(ByVal ReadAttachment)
    If ReadAttachment.Type = 1 then
        MsgBox "If you change this file, save your changes to the or
    End If
End Sub
```

BeforeAttachmentSave Event

Occurs just before an attachment is saved.

Sub object_BeforeAttachmentSave(ByVal Attachment As Attachment, Cancel As Boolean)

object An expression that evaluates to one of the objects in the Applies To list. In VBScript, use the word **Item**.

Attachment Required. The **Attachment** to be saved.

Cancel Optional (not used in VBScript). **False** when the event occurs. If the event procedure sets this argument to **True**, the save operation is not completed and the attachment is not changed.

Remarks

In VBScript, if you set the return value of this function to **False**, the save operation is cancelled and the attachment is not changed.

This VBScript example notifies the user that the user is not allowed to save the attachment. It returns **False** to cancel the save operation.

ction Item_ BeforeAttachmentS MsgBox "You are not allowed Item_BeforeAttachmentSave = Function	l to save " & SaveAttachment.Name	

BeforeCheckNames Event

Occurs just before Microsoft Outlook starts resolving names in the recipient collection for an item.

Sub object_**BeforeCheckNames(Cancel As Boolean)**

object An expression that evaluates to an object in the Applies To list. In VBScript, use the word **Item**.

Cancel Optional (not used in VBScript). **False** when the event occurs. If the event procedure sets this argument to **True**, the operation is cancelled and the names in the recipients collection are not resolved.

Remarks

In VBScript, if you set the return value of this function to **False**, the operation is cancelled and the names in the recipients collection are not resolved.

This VBScript example asks the user if the user wants to resolve names and returns **False** to cancel the operation if the user answers no.

```
Function Item_BeforeCheckNames()
    If MsgBox ("Do you want to resolve names now?",4) = 7 Then
        Item_BeforeCheckNames = False
    End If
End Function
```

BeforeDelete Event

Occurs before an item is deleted.

Sub expression_BeforeDelete(ByVal Item As Object, Cancel As Boolean)

expression An object in the Applies To list declared with events in a class module.

Item Required **Object**. The item being deleted.

Cancel Required **Boolean**. **False** when the event occurs. If the event procedure sets this argument to **True**, the operation is not completed and the item is not deleted.

Remarks

In order for this event to fire when a mail message, distribution list, journal entry, task, contact or post are deleted through an action, an Inspector must be open.

The event occurs each time an item is deleted.

The following example prompts the user regarding whether to delete the specified item. If the user clicks **No**, the item will not be deleted. If the user clicks **Yes**, the item will be deleted and a message will be displayed to the user.

Private Sub objMail_BeforeDelete(ByVal Item As Object, Cancel As Boo'Prompts the user before deleting an item

```
Dim strPrompt As String
   'Prompt the user for a response
   strPrompt = "Are you sure you want to delete the item?"
   If MsgBox(strPrompt, vbYesNo + vbQuestion) = vbNo Then
        'Don't delete the item
        Cancel = True
   Else
        MsgBox ("Item Deleted")
        'Delete the item
        Cancel = False
   End If
End Sub
```

BeforeFolderSwitch Event

Occurs before the explorer navigates to a new folder, either as a result of user action or through program code. This event is not available in VBScript.

Note If the folder being switched to is in a name space that doesn't support Automation (such as the file system), *NewFolder* is **Nothing**.

Sub object_BeforeFolderSwitch(ByVal NewFolder As Object, Cancel As Boolean)

object An expression that evaluates to an **Explorer** object.

NewFolder Required. The <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, and the Initialize_handler routine must be called before the event procedure can be called by Microsoft Outlook.

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

Public Sub Initialize_handler()
    Set myOlExp = myOlApp.ActiveExplorer

End Sub

Private Sub myOlExp_BeforeFolderSwitch(ByVal NewFolder As Object, Ca
    If NewFolder.Name = "Off Limits" Then
        MsgBox "You do not have permission to access this folder."
        Cancel = True
    End If
End Sub
```

▼ Show All

BeforeGroupAdd Event

-

Occurs before a new group is added to the <u>Outlook Bar</u>, either as a result of user action or through program code. This event is not available in VBScript.

Sub object_**BeforeGroupAdd(**Cancel **As Boolean)**

object An expression that evaluates to an <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 Outlook Bar.

This example prevents the user from adding a group to the Outlook Bar. The sample code must be placed in a class module, and the Initialize_handler routine must be called before the event procedure can be called by Microsoft Outlook.

```
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
```

▼ Show All

BeforeGroupRemove Event

Occurs before a new group is removed from the <u>Outlook Bar</u>, either as a result of user action or through program code. This event is not available in VBScript.

Sub object_BeforeGroupRemove(ByVal Group As OutlookBarGroup, Cancel As Boolean)

object An expression that evaluates to an <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 Outlook Bar.

This example prevents the user from removing a group from the Outlook Bar. The sample code must be placed in a class module, and the Initialize_handler routine must be called before the event procedure can be called by Microsoft Outlook.

```
Dim myOlApp As New Outlook.Application
Dim WithEvents myOlGroups As Outlook.OutlookBarGroups
Dim myOlBar As Outlook.OutlookBarPane

Sub Initialize_handler()
    Set myOlBar = myOlApp.ActiveExplorer.Panes.item("OutlookBar")
    Set myOlGroups = myOlBar.Contents.Groups
End Sub

Private Sub myOlGroups_BeforeGroupRemove(ByVal Group As OutlookBarGr Cancel = True
End Sub
```

▼ Show All

BeforeGroupSwitch Event

Occurs before a new group is opened in the <u>Outlook Bar</u>, either as a result of user action or through program code. This event is not available in VBScript.

Sub object_BeforeGroupSwitch(ByVal ToGroup As OutlookBarGroup, Cancel as Boolean)

object An expression that evaluates to an **OutlookBarPane** object.

ToGroup Required. The **OutlookBarGroup** being opened.

Cancel Optional. **False** when the event occurs. If the event procedure sets this argument to **True**, the change is cancelled and the current group remains open.

This example prevents the user from opening the Outlook Bar group named Other Shortcuts. The sample code must be placed in a class module, and the Initialize_handler routine must be called before the event procedure can be called by Microsoft Outlook.

```
Dim myOlApp As New Outlook.Application
Public WithEvents myOlPane As Outlook.OutlookBarPane

Public Sub Initialize_handler()
    Set myOlPane = myOlApp.ActiveExplorer.Panes.Item("OutlookBar")
End Sub

Private Sub myOlPane_BeforeGroupSwitch(ByVal ToGroup As Outlook.Outl
    If ToGroup.Name = "Other Shortcuts" Then Cancel = True
End Sub
```

BeforeItemCopy Event

Occurs when an item is copied. This event can be cancelled after it has started.

Private Sub expression_BeforeItemCopy(Cancel As Boolean)

expression An expression that returns an **Explorer** object declared with events in a class module.

Cancel Required **Boolean**. **False** when the event occurs. If the event procedure sets this argument to **True**, the operation is not completed and the item is not copied.

The following example prompts the user before an item is copied. A message is displayed to the user verifying that the item should be copied. If the user clicks **Yes,** the item is copied.

```
Private Sub objExplorer_BeforeItemCopy(Cancel As Boolean)
'Prompts the user before copying an item

Dim lngAns As Long 'user answer
'Display question to user
lngAns = MsgBox("Are you sure you want to copy the item?", 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 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.

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.

End Sub

The following example prompts the user before pasting the contents of the Clipboard to the specified target. If the user clicks **Yes**, the contents of the Clipboard are copied to the specified target destination and a message box is displayed for each item copied.

```
Private Sub objExplorer_BeforeItemPaste(ClipboardContent As Variant,
    Dim lngAns As Integer 'users' answer
    'Prompt user about paste
    lngAns = MsgBox("Are you sure you want to past the contents of t
                    & Target.Name & "?", vbYesNo)
    If lngAns = vbYes Then
    'If user wants to paste
        If TypeOf ClipboardContent Is Selection Then
            'if is collection, display each object name
            Dim obj As Object
            For Each obj In ClipboardContent
                'Display subject of item
                MsgBox "Pasting Item: " & obj.Subject
            'Next Item
            Next
        End If
        Cancel = False
    Else
        'If user clicks no, display message and cancel paste
        MsqBox "The clipboard content was not pasted."
        Cancel = True
    End If
```

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 example prompts the user with a warning message before maximizing the current window. If the user clicks **Yes**, the explorer will maximize.

```
Private Sub objExplorer_BeforeMaximize(Cancel As Boolean)
'Prompts the user before maximizing the explorer

Dim lngAns As Long

lngAns = MsgBox("Are you sure you want to maximize the current wif 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 object in the Applies To list declared with events in a class module.

Cancel Required **Boolean**. **False** when the event occurs. If the event procedure sets this argument to **True**, the operation is not completed and the Explorer or Inspector is not minimized.

The following example prompts the user with a message before the window is minimized. If the user clicks **Yes**, the Explorer is minimized.

```
Private Sub objExplorer_BeforeMinimize(Cancel As Boolean)

'Prompts the user before minimizing the Explorer

Dim lngAns As Long

lngAns = MsgBox("Are you sure you want to minimize the current will 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 example prompts the user with a message before the **Explorer** is moved by the user. If the user clicks **Yes**, the **Explorer** can be moved by the user.

```
Private Sub objExplorer_BeforeMove(Cancel As Boolean)
'Prompts the user before moving the window

Dim lngAns As Long

lngAns = MsgBox("Are you sure you want to move the current window

If lngAns = vbYes Then
Cancel = False

Else
Cancel = True

End If

End Sub
```

▼ Show All

BeforeNavigate Event

Occurs when the user clicks on an <u>Outlook Bar</u> shortcut to navigate to a different folder. This event is not available in VBScript.

Sub object_BeforeNavigate(ByVal Shortcut As OutlookBarShortcut, Cancel As Boolean)

object An expression that evaluates to an **OutlookBarPane** object.

Shortcut Required. The shortcut that the user clicked.

Cancel Optional. **False** when the event occurs. If the event procedure sets this argument to **True**, the current folder is not changed.

This example prevents the user from using the Outlook Bar to open the Notes folder. The sample code must be placed in a class module, and the Initialize_handler routine must be called before the event procedure can be called by Microsoft Outlook.

BeforeReminderShow Event

Occurs before the reminder dialog is displayed.

Private Sub expression_**BeforeReminderShow**(Cancel As Boolean)

expression A variable which references an object of type **Reminders** declared with events in a class module.

Cancel Required. **True** to cancel the event. The default value is **False**.

The following example prompts the user before displaying a reminder. The user can cancel the events by clicking **No**.

```
Private Sub objReminders_BeforeReminderShow(Cancel As Boolean)
'Occurs before a reminder appears

Dim lngAns As Long

lngAns = MsgBox("Do you want to view the reminder?", vbYesNo)
If lngAns = vbYes Then
        Cancel = False
Else
        Cancel = True
End If

End Sub
```

▼ Show All

BeforeShortcutAdd Event

_

Occurs before a new <u>shortcut</u> is added to a group in the <u>Outlook Bar</u>, either as a result of user action or through program code. This event is not available in VBScript.

Sub object_BeforeShortcutAdd(Cancel As Boolean)

object An expression that evaluates to an <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 example prevents a user from adding a shortcut to the first Outlook Bar group. The sample code must be placed in a class module, and the Initialize_handler routine must be called before the event procedure can be called by Microsoft Outlook.

```
Dim myOlApp As New Outlook.Application
Dim WithEvents myOlShortcuts As Outlook.OutlookBarShortcuts
Dim myOlBar As Outlook.OutlookBarPane

Sub Initialize_handler()
    Set myOlBar = myOlApp.ActiveExplorer.Panes.Item("OutlookBar")
    Set myOlShortcuts = myOlBar.Contents.Groups.Item(1).Shortcuts
End Sub

Private Sub myOlShortcuts_BeforeShortcutAdd(Cancel As Boolean)
    MsgBox "You are not allowed to add a shortcut to this group."
    Cancel = True
End Sub
```

▼ Show All

BeforeShortcutRemove Event

Occurs before a new <u>shortcut</u> is removed from a group in the <u>Outlook Bar</u>, either as a result of user action or through program code. This event is not available in VBScript.

Sub object_BeforeShortcutRemove(ByVal Shortcut As OutlookBarShortcut, Cancel As Boolean)

object An expression that evaluates to an <u>OutlookBarShortcuts</u> collection object.

Shortcut Required. The **OutlookBarShortcut** that is being removed.

Cancel Optional. **False** when the event occurs. If the event procedure sets this argument to **True**, the shortcut is not removed from the group.

The following example prevents a user from removing a shortcut from the first Outlook Bar group. The sample code must be placed in a class module, and the Initialize_handler routine must be called before the event procedure can be called by Microsoft Outlook.

```
Dim myOlApp As New Outlook.Application
Dim WithEvents myOlShortcuts As Outlook.OutlookBarShortcuts
Dim myOlBar As Outlook.OutlookBarPane

Sub Initialize_handler()
    Set myOlBar = myOlApp.ActiveExplorer.Panes.Item("OutlookBar")
    Set myOlShortcuts = myOlBar.Contents.Groups.Item(1).Shortcuts
End Sub

Private Sub myOlShortcuts_BeforeShortcutRemove(Cancel As Boolean)
    MsgBox "You are not allowed to remove a shortcut from this group Cancel = True
End Sub
```

BeforeSize Event

Occurs when the user sizes the current **Explorer** or **Inspector**. This event can be cancelled after it has started. If the event is cancelled, the window is not sized.

Private Sub *expression_***BeforeSize**(*Cancel* **As Boolean**)

expression An expression that returns an object in the Applies To list declared with events in a class module.

Cancel Required **Boolean**. **False** when the event occurs. If the event procedure sets this argument to **True**, the operation is not completed and the **Explorer** or **Inspector** is not sized.

The following example prompts the user with a warning message before the Inspector is sized. If the user clicks **Yes**, the Inspector can be sized.

▼ Show All

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 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 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.

▼ Show All

Close Event

Occurs when the **Inspector** associated with an Outlook <u>item</u> is being closed.

Sub object_Close(Cancel As Boolean)

object An expression that evaluates to an object in the Applies To list. In VBScript, use the word **Item**.

Cancel Optional (not used in VBScript). **False** when the event occurs. If the event procedure sets this argument to **True**, the close operation is not completed and the inspector is left open.

Remarks

In VBScript, if you set the return value of this function to **False**, the close operation isn't completed and the inspector is left open. This event cannot be cancelled for the **Inspector** and **Explorer** objects.

If you use the <u>Close</u> method to fire this event, it can only be canceled if the Close method uses the *olPromptForSave* argument.

This 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.

```
Function Item_Close()
   If Not Item.Saved Then
        Item.Save
   End If
End Function
```

▼ Show All

CustomAction Event

Occurs when a custom action of an 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 VBScript, if you set the return value of this function to **False**, the custom action operation is not completed.

This example uses the **CustomAction** event to set a property on the response item.

```
Function Item_CustomAction(ByVal myAction, ByVal myResponse)
Select Case myAction.Name
Case "Action1"
myResponse.Subject = "Changed by VB Script"
Case Else
End Select
End Function
```

▼ Show All

CustomPropertyChange Event

Occurs when a custom property of an 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 object in the Applies To list. In VBScript, use the word **Item**.

Name Required. The name of the custom property that was changed.

This VBScript example uses the **CustomPropertyChange** event to enable a control when a Boolean field is set to **True**.

For this example, create two custom fields on the second page of a <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 ceases to be the active window, either as a result of user action or through program code. This event is not available in VBScript.

Sub object_Deactivate()

object An expression that evaluates to an **Explorer** or **Inspector** object.

This 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 Outlook.Application
Public WithEvents myOlExp As Outlook.Explorer

Public Sub Initialize_handler()
    Set myOlExp = myOlApp.ActiveExplorer
End Sub

Private Sub myOlExp_Deactivate()
    myOlExp.WindowState = olMinimized
End Sub
```

FolderAdd Event

Occurs when a folder is added to the specified **Folders** collection. This event is not available in VBScript.

Sub object_FolderAdd(ByVal Folder As MAPIFolder)

object An expression that evaluates to a **Folders** collection object.

Folder Required. The <u>MAPIFolder</u> that was added to the collection.

This example displays a new folder created in the user's Inbox folder.

```
Dim myOlApp As New Outlook.Application
Public WithEvents myOlFolders As Outlook.Folders

Public Sub Initialize_handler()
    Set myOlFolders = myOlApp.GetNamespace("MAPI").GetDefaultFolder(
End Sub

Private Sub myOlFolders_FolderAdd(ByVal Folder As Outlook.MAPIFolder
    Folder.Display
End Sub
```

FolderChange Event

_

Occurs when a folder in the specified **Folders** collection is changed. This event is not available in VBScript.

Sub object_FolderChange(ByVal Folder As MAPIFolder)

object An expression that evaluates to a **Folders** collection object.

Folder Required. The **MAPIFolder** that was changed.

This example prompts the user to remove a folder from the Deleted Items folder if the folder is empty. The sample code must be placed in a class module, and the Initialize_handler routine must be called before the event procedure can be called by Microsoft Outlook.

FolderRemove Event

Occurs when a folder is removed from the specified <u>Folders</u> collection. This event is not available in VBScript.

Sub object_**FolderRemove()**

object An expression that evaluates to a **Folders** collection object.

This example fills a combo box on a form with the names of the folders in the Deleted Items folder. The sample code must be placed in a class module, and the Initialize_handler routine must be called before the event procedure can be called by Microsoft Outlook.

```
Dim myolapp As New Outlook.Application
Dim WithEvents myFolders As Outlook.Folders

Sub Initialize_handler()
    Set myNS = myolapp.GetNamespace("MAPI")
    Set myFolders = myNS.GetDefaultFolder(olFolderDeletedItems).Fold

End Sub

Private Sub myFolders_FolderRemove()
    Form1.Combo1.Clear
    For x = 1 To myFolders.Count
        Form1.Combo1.AddItem (myFolders.Item(x).Name)
    Next x

End Sub

Sub Title (x) Name)
```

FolderSwitch Event

_

Occurs when the explorer navigates to a new folder, either as a result of user action or through program code. This event is not available in VBScript.

Sub object_FolderSwitch()

object An expression that evaluates to an **Explorer** object.

This example shows or hides a menu bar, depending on the folder that is visible. The sample code must be placed in a class module, and the Initialize_handler routine must be called before the event procedure can be called by Microsoft Outlook.

▼ Show All

Forward Event

Occurs when the user selects the **Forward** action for an 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 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 example uses the **Forward** event to disable forwarding on an <u>item</u> by setting the return value to **False** and displays a message that the item may not be forwarded.

▼ Show All

GroupAdd Event

-

Occurs when a new group has been added to the <u>Outlook Bar</u>. This event is not available in VBScript.

Sub object_GroupAdd(ByVal NewGroup As OutlookBarGroup)

object An expression that evaluates to an **OutlookBarGroups** object.

NewGroup Required. The <u>OutlookBarGroup</u> that was added.

This example adds a shortcut to a group when the group is created. The sample code must be placed in a class module, and the Initialize_handler routine must be called before the event procedure can be called by Microsoft Outlook.

ItemAdd Event

Occurs when an item is added to the specified collection. This event is not available in VBScript.

Sub object_ItemAdd(ByVal Item As Object)

object An expression that evaluates to one of the objects in the Applies To list.

Item Required. The item that was added.

In this example, when a new contact is added to the Contacts folder, the contact item is attached to a mail message and sent to a distribution list named Sales Team. The sample code must be placed in a class module, and the Initialize_handler routine must be called before the event procedure can be called by Microsoft Outlook.

```
Dim myOlApp As Outlook.Application
Public WithEvents myOlItems As Outlook. Items
Public Sub Initialize_handler()
    Set myOlItems = myOlApp.GetNamespace("MAPI").GetDefaultFolder(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 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.

ItemRemove Event

Occurs when an item is deleted from the specified collection. This event is not available in VBScript.

Sub object_ItemRemove()

object An expression that evaluates to one of the objects in the Applies To list.

This example optionally sends a notification message to a workgroup when the user removes a contact from the default Contacts folder. The sample code must be placed in a class module, and the Initialize_handler routine must be called before the event procedure can be called by Microsoft Outlook.

```
Dim myOlApp As Outlook.Application
Public WithEvents myOlItems As Outlook.Items

Public Sub Initialize_handler()
    Set myOlItems = myOlApp.GetNamespace("MAPI").GetDefaultFolder(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 = "Please remove the following contact from y myOlMItem.Display
    End If
End Sub
```

ItemSend Event

-

Occurs whenever an item is sent, either by the user through an **Inspector** (before the inspector is closed, but after the user clicks the **Send** button) or when the **Send** method is used in a program. This event is not available in VBScript.

Sub object_ItemSend(ByVal Item As Object, Cancel As Boolean)

object An expression that evaluates to an **Application** object.

Item Required. The item being sent.

Cancel Optional. **False** when the event occurs. If the event procedure sets this argument to **True**, the send action is not completed and the inspector is left open.

The following example shows how to cancel the <u>ItemSend</u> event in response to user input.

```
Private Sub myOlApp_ItemSend(ByVal Item As Object, Cancel As Boolean
    Prompt$ = "Are you sure you want to send " & Item.Subject & "?"
    If MsgBox(Prompt$, vbYesNo + vbQuestion, "Sample") = vbNo Then
        Cancel = True
    End If
End Sub
```

MapiLogonComplete Event

Occurs after the user has logged onto the system.

Private Sub expression_MapiLogonComplete()

expression A variable which references an object of type **Application** declared with events in a class module.

The following example displays a message after the user has logged onto the system.

```
Private Sub objApp_MapiLogonComplete()
'Occurs when a user has logged on

MsgBox "Logon complete"

End Sub
```

NewExplorer Event

Occurs whenever a new explorer window is opened, either as a result of user action or through program code. This event is not available in VBScript.

Sub object_NewExplorer(ByVal Explorer As Explorer)

object An expression that evaluates to an **Explorers** collection object.

Explorer Required. The explorer that was opened.

Remarks

The event occurs after the new **Explorer** object is created but before the explorer window appears.

This example minimizes the currently active explorer window when a new explorer is about to appear. The sample code must be placed in a class module, and the Initialize_handler routine must be called before the event procedure can be called by Microsoft Outlook.

```
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 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 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 messages are received in the Inbox. This event is not available in VBScript.

Sub object_NewMail()

object An expression that evaluates to an **Application** object.

This example displays the Inbox folder when new mail arrives. The sample code must be placed in a class module, and the Initialize_handler routine must be called before the event procedure can be called by Microsoft Outlook.

```
Dim WithEvents myOlApp As Outlook.Application
Sub Initialize handler()
    Set myOlApp = CreateObject("Outlook.application")
End Sub
Private Sub myOlApp_NewMail()
    Dim myExplorers As Outlook.Explorers
    Dim myFolder As Outlook.MAPIFolder
    Set myExplorers = myOlApp.Explorers
    Set myFolder = myOlApp.GetNamespace("MAPI").GetDefaultFolder(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
```

OnError Event

Occurs when Microsoft Outlook encounters an error while synchronizing a user's folders using the specified synchronization profile. This event is not available in VBScript.

Sub object_OnError(ByVal Code As Long, ByVal Description As String)

object An expression that evaluates to a **SyncObject** object.

Code A unique value that identifies the error.

Description Required. A textual description of the error.

This example displays a message box describing the synchronization error and sets attributes of controls on a form. The sample code must be placed in a class module, and the Initialize_handler routine must be called before the event procedure can be called by Microsoft Outlook.

▼ Show All

Open Event

Occurs when a Microsoft Outlook <u>item</u> is being opened in an <u>Inspector</u>. When this event occurs, the <u>Inspector</u> 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 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 VBScript example uses the **Open** event to display the "All Fields" page every time the <u>item</u> is opened.

```
Function Item_Open()
    Item.GetInspector.SetCurrentFormPage "All Fields"
End Function
```

This VBScript example uses the <u>Unread</u> 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.

OptionsPagesAdd Event

Occurs whenever the **Options** dialog box (available through the **Tools** menu) or a folder **Properties** dialog box is opened. This event is not available in VBScript.

Sub object_OptionsPagesAdd(ByVal Pages As PropertyPages, ByVal Folder As MAPIFolder)

object An expression that evaluates to an **Application** or a **NameSpace** object.

Pages Required. The collection of property pages that have been added to the dialog box. This collection includes only custom property pages. It does not include standard Microsoft Outlook property pages.

Folder This argument is only used with the <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 example adds a new property page to the Microsoft Outlook **Options** dialog box. The sample code must be placed in a class module of a <u>Common Object Model (COM) add-in</u>.

Progress Event

Occurs periodically while Microsoft Outlook is synchronizing a user's folders using the specified synchronization profile. This event is not available in VBScript.

Sub object_Progress(ByVal State As OlSyncState, ByVal Description As String, ByVal Value As Long, ByVal Max As Long)

object An expression that evaluates to a **SyncObject** object.

State Required. A value that identifies the current state of the synchronization process. Can be either of the following **OlSyncState** constants: **olSyncStarted** or **olSyncStopped**.

Description Required. A textual description of the current state of the synchronization process.

Value Required. Specifies the current value of the synchronization process (such as the number of items synchronized).

Max Required. The maximum that *Value* can reach. The ratio of *Value* to *Max* represents the percent complete of the synchronization process.

This example updates a label on a form to show the progress of synchronization. The sample code must be placed in a class module, and the Initialize_handler routine must be called before the event procedure can be called by Microsoft Outlook.

```
Dim myOlApp As New Outlook.Application
Dim WithEvents mySync As Outlook.SyncObject
Dim myForm As New Form1

Sub Initialize_handler()
    Set mySync = myOlApp.Session.SyncObjects.Item(1)
End Sub

Private Sub mySync_Progress(ByVal State As Outlook.OlSyncState, ByVa
    If State = olSyncStarted then
        Cap = "Synchronization started: "
    Else
        Cap = "Synchronization stopped: "
    End If
    Cap = Cap & Str(State / Max * 100) & "% " & Description
    Form1.Label1.Caption = Cap
End Sub
```

▼ Show All

PropertyChange Event

Occurs when a standard property (for example, Subject or To) of a Microsoft Outlook <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 VBScript, use the word **Item**.

Name Required. The name of the property that was changed.

This example uses the **PropertyChange** event to disable the setting of a reminder on an item.

Quit Event

Occurs when Microsoft Outlook begins to close. This event is not available in VBScript.

Sub object_Quit()

object An expression that evaluates to an **Application** object.

This example displays a farewell message when Microsoft Outlook exits. The sample code must be placed in a class module, and the Initialize_handler routine must be called before the event procedure can be called by Outlook.

```
Dim WithEvents myOlApp As Outlook.Application

Sub Initialize_handler()
    Set myOlApp = CreateObject("Outlook.application")
End Sub

Private Sub myOlApp_Quit()
    MsgBox "Goodbye, " & Application.GetNamespace("MAPI").CurrentUse
End Sub
```

▼ Show All

Read Event

_

Occurs when an existing Microsoft Outlook <u>item</u> is opened for editing by the user. The **Read** event differs from the <u>Open</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 <u>Inspector</u>.

Sub object_Read()

object

An object that evaluates to one of the objects in the Applies To list. In VBScript, use the word **Item**.

This VBScript example uses the **Read** event to increment a counter that tracks how often an <u>item</u> is read.

```
Sub Item_Read()
    Set myProperty = Item.UserProperties("ReadCount")
    myProperty.Value = myProperty.Value + 1
    Item.Save
End Sub
```

▼ Show All

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 **Application** object.

Item The **AppointmentItem**, **MailItem**, **ContactItem** or **TaskItem** associated with the reminder. If the appointment associated with the reminder is a recurring appointment, **Item** is the specific occurrence of the appointment that displayed the reminder, not the master appointment.

This Microsoft Visual Basic/Visual Basic for Applications example tests the item generating the reminder to determine if it is a mail item. If it is, the example uses the **ReplyAll** method to create and display a new mail item. The sample code must be placed in a class module, and the Initialize_handler routine must be called before the event procedure can be called by Microsoft Outlook.

```
Dim WithEvents myolapp As Outlook.Application

Sub Initialize_handler()
    Set myolapp = CreateObject("Outlook.application")

End Sub

Private Sub myolapp_Reminder(ByVal Item As Object)
    Dim myReplyItem As Outlook.MailItem
    If TypeName(Item) = "MailItem" Then
        Set myReplyItem = Item.ReplyAll
        myReplyItem.Display
    End If

End Sub
```

ReminderAdd Event

Occurs after a reminder is added.

Private Sub expression_ReminderAdd(ByVal ReminderObject As Reminder)

expression A variable which references an object of type **Reminders** 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 tl	ie name	of the	new	reminder	when a	ı remin	der
is added to the	collection	1.							

Private Sub objReminders_ReminderAdd(ByVal ReminderObject As Reminde 'Occurs when a Reminder object is added to the collection

MsgBox "The following reminder has been added: " & _ ReminderObject.Caption

End Sub		

ReminderChange Event

Occurs after a reminder has been modified.

Private Sub *expression_***ReminderChange(ByVal** *ReminderObject* **As Reminder)**

expression A variable which references an object of type **Reminders** declared with events in a class module.

ReminderObject Required. The **Reminder** object that has been modified.

The following	example	prompts	the user	with a	message	every t	ime a	reminder
is modified.	_				_	-		

Private Sub objReminders_**ReminderChange**(ByVal ReminderObject As Remi'Occurs when reminder is changed

MsgBox "The reminder " & ReminderObject.Caption & " has changed.

End Sub

ReminderFire Event

Occurs before the reminder is executed.

Private Sub expression_**ReminderFire(ByVal** ReminderObject As Reminder)

expression A variable which references an object of type **Reminders** declared with events in a class module.

ReminderObject Required. The **Reminder** object that has been executed.

The following	example of	displays a	n message to	the user	every	time a	reminde	r is
executed.								

Private Sub objReminders_**ReminderFire**(ByVal ReminderObject As Remind 'Occurs when a reminder executes

	MsgBox	"The	reminder	11	&	ReminderObject	.Caption	&	11	has	just	exe
End	Sub											

ReminderRemove Event

Occurs when a **Reminder** object has been removed from the collection.

Private Sub expression_ReminderRemove()

expression A variable which references an object of type **Reminders** declared with events in a class module.

Remarks

A reminder can be removed from the **Reminders** collection by any of the following means:

- The **Reminders** collection's **Remove** method.
- The **Reminder** object's **Dismiss** method.
- When the user clicks the **Dismiss** button.
- When a user turns off a meeting reminder from within the associated item.
- When a user deletes an item that contains a reminder.

The following example displays a message to the user when a **Reminder** object is removed from the collection.

'0cc	vate Sub objReminders_ ReminderRemove() curs when a reminder is removed from the collection the user clicks Dismiss
	MsgBox "A reminder has been removed from the collection."
End	Sub

▼ Show All

Reply Event

Occurs when the user selects the **Reply** action for an 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 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 VBScript 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.

	em_ Reply (esponse.S n	-	-	-	Item.Par	ent	

▼ Show All

ReplyAll Event

Occurs when the user selects the **ReplyAll** action for an 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 VBScript 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.

```
Function Item_ReplyAll(ByVal myResponse)
    myMsg = "Do you really want to reply to all original recipients?
myResult = MsgBox(myMsg, 289, "Flame Protector")
    If myResult = 1 Then
        Item_ReplyAll = True
    Else
        Item_ReplyAll = False
    End If
End Function
```

SelectionChange Event

Occurs when the selection of the current view changes. Other selection changes (such as the selected folder) do not cause this event to occur. In addition, this event does not occur if the current folder is a file-system folder or if Outlook Today or any folder with a current Web view is displayed. This event is not available in VBScript.

Sub object_**SelectionChange()**

object An expression that evaluates to an **Explorer** object.

This example changes the caption of a form named Form1 to show the number of items selected in the topmost explorer window. The sample code must be placed in a class module, and the Initialize_handler routine must be called before the event procedure can be called by Microsoft Outlook.

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

Public Sub Initialize_handler()
        Set myOlExp = myOlApp.ActiveExplorer
End Sub

Private Sub myOlExp_SelectionChange()
        Form1.Caption = myOlExp.Selection.Count & " items selected."
End Sub
```

▼ Show All

Send Event

Occurs when the user selects the **Send** 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 VBScript, if you set the return value of this function to **False**, the item is not sent.

This VBScript example uses the <u>Send</u> event and sends an <u>item</u> with an automatic expiration date of one week.

```
Function Item_Send()
   Item.ExpiryTime = Date + 7
End Function
```

▼ Show All

ShortcutAdd Event

-

Occurs when a new <u>shortcut</u> is added to a <u>Outlook Bar</u> group. This event is not available in 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 example changes the name of a Calendar shortcut when it is added to the first group in the Outlook Bar. The sample code must be placed in a class module, and the Initialize_handler routine must be called before the event procedure can be called by Microsoft Outlook.

```
Dim myOlApp As New Outlook.Application
Dim WithEvents myOlSCuts As Outlook.OutlookBarShortcuts
Dim myOlBar As Outlook.OutlookBarPane

Sub Initialize_handler()
    Set myOlBar = myOlApp.ActiveExplorer.Panes.Item("OutlookBar")
    Set myOlSCuts = myOlBar.Contents.Groups.Item(1).Shortcuts

End Sub

Private Sub myOlSCuts_ShortcutAdd(ByVal NewShortcut As Outlook.Outlo
    Dim myNS As Outlook.NameSpace
    Set myNS = myOlApp.GetNamespace("MAPI")
    If NewShortcut.Target.Name = "Calendar" Then
        NewShortcut.Name = myNS.CurrentUser & "'s Schedule"
    End If
End Sub
```

Snooze Event

Occurs when a reminder is dismissed using the snooze button.

Private Sub expression_Snooze(ByVal ReminderObject As Reminder)

expression A variable which references an object of type **Reminders** declared with events in a class module.

Remarks

This event will fire when the **Snooze** method is executed, or when the user clicks the **Snooze** button.

The following example displays the caption of the **Reminder** object that has been snoozed.

Startup Event

Occurs when Microsoft Outlook is starting, but after all add-in programs have been loaded. This event is not available in VBScript.

Sub object_Startup()

object An expression that evaluates to an **Application** object.

Remarks

An Outlook Visual Basic for Applications macro can use this event procedure to initialize itself when Outlook starts.

This Microsoft Outlook Visual Basic for Applications example maximizes the Outlook explorer window when Outlook starts.

<pre>Private Sub Application_Startup() Application.ActiveExplorer.WindowState = olMaximized End Sub</pre>									

SyncEnd Event

_

Occurs immediately after Microsoft Outlook finishes synchronizing a user's folders using the specified synchronization profile. This event is not available in VBScript.

Sub object_SyncEnd()

object An expression that evaluates to a **SyncObject** object.

This example updates a label on a form to indicate that synchronization has finished and changes the enabled state of buttons on the form. The sample code must be placed in a class module, and the Initialize_handler routine must be called before the event procedure can be called by Microsoft Outlook.

```
Dim myOlApp As New Outlook.Application
Dim WithEvents mySync As Outlook.SyncObject
Dim myForm As New Form1

Sub Initialize_handler()
   Set mySync = myOlApp.Session.SyncObjects.Item(1)
End Sub

Private Sub mySync_SyncEnd()
   Form1.Label1.Caption = "Synchronization complete."
   Form1.Command1.Enabled = True
   Form1.Command2.Enabled = False
End Sub
```

SyncStart Event

-

Occurs when Microsoft Outlook begins synchronizing a user's folders using the specified synchronization profile. This event is not available in VBScript.

Sub object_SyncStart()

object An expression that evaluates to a **SyncObject** object.

This example updates a label on a form to show that synchronization has started. The sample code must be placed in a class module, and the Initialize_handler routine must be called before the event procedure can be called by Microsoft Outlook.

```
Dim myOlApp As New Outlook.Application
Dim WithEvents mySync As Outlook.SyncObject
Dim myForm As New Form1

Sub Initialize_handler()
    Set mySync = myOlApp.Session.SyncObjects.Item(1)
End Sub

Private Sub mySync_SyncStart()
    Form1.Label1.Caption = "Synchronization started"
End Sub

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 example displays the view's name and saves it. Use the <u>Save</u> method after the properties have been modified to save the changes to the view.

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 example displays the name of the view that has been removed from the collection.

ViewSwitch Event

-

Occurs when the view in the explorer changes, either as a result of user action or through program code. This event is not available in VBScript.

Sub object_ViewSwitch()

object An expression that evaluates to an **Explorer** object.

This example hides the preview pane if it is visible when the user switches to the Messages with AutoPreview view. The sample code must be placed in a class module, and the Initialize_handler routine must be called before the event procedure can be called by Microsoft Outlook.

```
Dim myolapp As New Outlook.Application
Dim WithEvents myOlExpl As Outlook.Explorer

Sub Initialize_handler()
    Set myOlExpl = myolapp.ActiveExplorer

End Sub

Private Sub myOlExpl_ViewSwitch()
    If myOlExpl.CurrentView = "Messages with AutoPreview" And myOlEx myOlExpl.ShowPane olPreview, False
    End If
End Sub
```

▼ Show All

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 VBScript, if you set the return value of this function to **False**, the save operation is not completed.

This VBScript example uses the **Write** 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.

```
Function Item_Write()
    myMsg = "The item is about to be saved. Do you wish to overwrite
    myResult = MsgBox(myMsg, 289, "Save")
    If myResult = 1 Then
        Item_Write = True
    Else
        Item_Write = False
    End If
End Function
```

Using events with Automation

To create an event handler for Microsoft Outlook objects in Microsoft Visual Basic or Microsoft Visual Basic for Applications in another application, you need to complete the following four steps:

- 1. Set a reference to the Microsoft Outlook Object Library.
- 2. Declare an object variable to respond to the events.
- 3. Write the specific event procedures.
- 4. Initialize the declared object.

Learn about working with events in Outlook Visual Basic for Applications.

Set the Reference to the Outlook Object Library

Before you can use an Outlook object in Visual Basic or Visual Basic for Applications code, you must first set a reference to the Outlook Object Model in the **References** dialog box. For more information about using this dialog box, see the online Help for your programming environment.

Declare the Object Variable

Once you've referenced the object model library, you must declare variables that reference the object you want to use. You can declare the variable in the module in which the object will be used (that is, the module containing the event-handler procedure), but more commonly you'll declare it in a class module so it can be used in any module in your program.

For example, to declare an object variable for the **Application** object in a class module, you use code like the following.

Public WithEvents myOlApp As Outlook.Application

You must use the **WithEvents** keyword to specify that the object variable will be used to respond to events triggered by the object.

Write the Event Procedure

After the new object has been declared with events, it appears in the **Object** list in the class module Code window, and you can select the object's event procedures from the **Procedures/Events** list. For example, when you select the **ItemSend** event for an **Application** object declared as myOlApp, the following empty procedure appears in the Code window.

Private Sub myOlApp_ItemSend(Item as Object, Cancel as Boolean)
End Sub

Initialize the Declared Object

Before the procedure will run, you must connect the declared object (in this example, myOlApp) with the **Application** object. If you declared the object in a class module named EventClassModule, then you can use the following code in any module.

Dim myClass as New EventClassModule
Sub Register_Event_Handler()
<pre>Set myClass.myOlApp = CreateObject("Outlook.Application")</pre>
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 ' It can be

ext_dm_HostShutdown Add-in was disconnected when Outlook

ext_dm_UserClosed ' ' ' ' ' ' ' ' The custom argument is ignor End Sub	closed. Add-in was disconnected when the user cleared the corresponding check box i COM Add-ins dialog box, or the Connec property of the corresponding COMAddI object was set to False.
' If the COM add-in connects a ' its startup routines. This e	OnStartupComplete(custom() As Variant) at startup, occurs when Outlook complet event does not occur if the COM add-ins, even when the user connects the addred.

Outlook Item Objects

Outlook items include

- AppointmentItem
- **ContactItem**
- DistListItem
- <u>DocumentItem</u>
- JournalItem
- MailItem
- MeetingItem
- NoteItem
- PostItem
- RemoteItem
- ReportItem
- TaskItem
- TaskRequestAcceptItem
- TaskRequestDeclineItem
- <u>TaskRequestItem</u>
- TaskRequestUpdateItem

GetNameSpace Method

Returns a **NameSpace** object of the specified type.

expression.GetNameSpace(Type)

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

Type Required **String**. The type of name space to return.

Remarks

The only supported name space type is "MAPI".

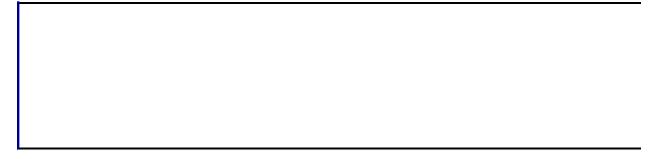
Example

This Visual Basic for Applications example uses **GetNameSpace** to obtain the MAPI **NameSpace** object.

```
Set myOlApp = CreateObject ("Outlook.Application")
Set myNameSpace = myOlApp.GetNameSpace("MAPI")
```

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

Set myNameSpace = Application.GetNameSpace("MAPI")



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.

CheckBoxForms.CheckBox.1ComboBoxForms.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

ScrollBarForms.ScrollBar.1SpinButtonForms.SpinButton.1TabStripForms.TabStrip.1TextBoxForms.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.

Application Access.Application

CurrentData Access.CodeData, Access.CurrentData

CurrentProject Access.CodeProject, Access.CurrentProject

DefaultWebOptions 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.

To create this object	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.

Application MSGraph.Application

Chart MSGraph.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.

ChartSpace OWC.Chart

DataSourceControl OWC.DataSourceControl

ExpandControl OWC.ExpandControl

PivotTable OWC.PivotTable

 $\textbf{RecordNavigationControl} \quad \text{OWC.} Record Navigation Control$

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