OLE Messaging Help

Overview of OLE Messaging

Programmer's Guide

Supplemental Information

Objects

Methods Properties Reference Summary

- All OLE Messaging Objects
- AddressEntry object
- Attachment object
- Attachments collection object
- Field object
- Fields collection object
- Folder object
- Folders collection object
- Message object
- Messages collection object
- Recipient object
- Recipients collection object
- Session object

Methods

Properties

Programmer's Reference Summary

- A -

- B -

- Z -

Introduction

The Microsoft OLE Messaging Library exposes messaging objects for use by Microsoft® Visual Basic® and Microsoft Visual C++™ applications.

The messaging library lets you quickly and easily add to your Visual Basic application the ability to send and receive mail messages and to interact with folders and address books. You can create programmable messaging objects, then use their properties and methods to meet the needs of your application.

When you combine messaging objects with other programmable objects exposed by Microsoft Access, Microsoft Excel, and Microsoft Word, you can build custom applications that, for example, would allow your users to extract information from a database, copy it to a spreadsheet for analysis, then create a report and mail the report to several people.

With these powerful building blocks, you can quickly build custom applications that cover your entire line-of-business needs.

The Microsoft OLE Messaging Library does not represent a new messaging model. It represents an additional interface to the Microsoft Messaging API (MAPI) model, designed to handle the most common tasks for client developers using Visual Basic and Visual C++.

This document assumes that you are familiar with the Microsoft Visual Basic programming model. To help you use the OLE Messaging Library, this document provides a short overview of the MAPI architecture. For complete reference information, see the *MAPI Programmer's Reference*.

The Microsoft OLE Messaging Library requires installation of MAPI 1.0 and a tool that supports OLE Automation. OLE Automation is supported by the following Microsoft applications:

- Microsoft Visual Basic version 3.0 or later
- Microsoft Visual Basic for Applications
- Microsoft Access version 2.0 or later
- Microsoft Excel version 5.0 or later
- Microsoft Project version 4.0 or later
- Microsoft Visual C++ version 1.5 or later

Note Microsoft Visual Basic 3.0 does not support multivalued properties.

Note This document describes Version 1.0 of the OLE Messaging Library. Microsoft is currently planning the next version of the OLE Messaging Library. According to the current design plans, the next version retains all of the functionality of Version 1.0 and adds several more messaging objects and collection objects. For more information, contact Microsoft.

Quick Start

The following sample program demonstrates how easy it is to add messaging to your applications when you use Visual Basic or Visual Basic for Applications.

In this example, we first create a Session object and log on. We then create a Message object and set its properties to indicate the message recipient, its subject, and the content of the message. We then call the Message object's **Send** method to transmit the message:

```
Function QuickStart()
Dim objSession As Object    ' Session object
Dim objMessage As Object    ' Message object
Dim objOneRecip As Object   ' Recipient object

    On Error GoTo error_olemsg

    ' create a session then log on, supplying username and password
    Set objSession = CreateObject("MAPI.Session")
    ' change the parameters to valid values for your configuration
    objSession.Logon 'profileName:="Princess Leia"

    ' create a message and fill in its properties
    Set objMessage = objSession.Outbox.Messages.Add
    objMessage.Subject = "Gift of droids"
    objMessage.Text = "Help us, Obi-wan. You are our only hope."

    ' create the recipient
    Set objOneRecip = objMessage.Recipients.Add
    objOneRecip.Name = "Obi-wan Kenobi"
    objOneRecip.Type = mapiTo
    objOneRecip.Resolve

    ' send the message and log off
    objMessage.Update
    objMessage.Send showDialog:=False
    MsgBox "The message has been sent"
    objSession.Logoff
    Exit Function

error_olemsg:
    MsgBox "Error " & Str(Err) & ": " & Error$(Err)
    Exit Function

End Function
```

The OLE Messaging Library invalidates the Message object after you call its **Send** method. In this example, the developer's code logs off to end the session after sending the message, but if you were to continue the MAPI session, you could avoid potential errors by setting the Message object to "Nothing" .

About This Document

Now that you've seen the power of the OLE Messaging Library, this document explains how you can use it in your own applications.

The section, "A Short Tour," defines the MAPI 1.0 terms used in this document and compares the OLE Messaging Library with the other MAPI programming interfaces. It then describes the design of the OLE Messaging Library, defining the objects and the collections of objects that are available to you with the OLE Messaging Library. This section also explains the relationships between these objects.

The section, "Programmer's Guide," offers sample Visual Basic code for many common programming tasks, such as creating and sending a message, posting a message to a public folder, traversing through folders, searching through address books, and handling errors.

The section, "Programmer's Reference," contains comprehensive reference information for the properties and methods of all objects and collection objects.

The appendixes offer additional background information about OLE Automation, the technology used by the OLE Messaging Library.

The best way to learn about OLE Messaging is to intersperse your reading with hands-on programming. You can use the sample code that is provided with the OLE Messaging Library. For information about the sample code, see the Release Notes.

A Short Tour

This chapter offers a whirlwind tour of MAPI 1.0 and describes how the OLE Messaging Library fits into the mix of MAPI programming interfaces. It provides a short description of OLE Automation, which is the basis of the design of the OLE Messaging Library. The chapter concludes with a conceptual overview of the OLE Messaging Library.

A Short Tour of Microsoft MAPI 1.0

Microsoft MAPI 1.0 defines a complete architecture for messaging applications. The architecture specifies several well-defined components. This allows system administrators to mix and match components to support a broad range of vendors, computing devices, and communication protocols.

This allows the MAPI architecture to be used for e-mail, scheduling, personal information managers, bulletin boards, and online services that run on mainframes, personal computers, and hand-held computing devices. The comprehensive architectural design allows MAPI to serve as the basis for a common information exchange.

MAPI programming interfaces allow access to all components or selected components of the architecture. The programming interface that allows access to all of the features of MAPI is known as *Extended MAPI*.

The MAPI architecture defines messaging *clients* that interact with various messaging *services* through the MAPI programming interfaces, as shown in the following diagram:

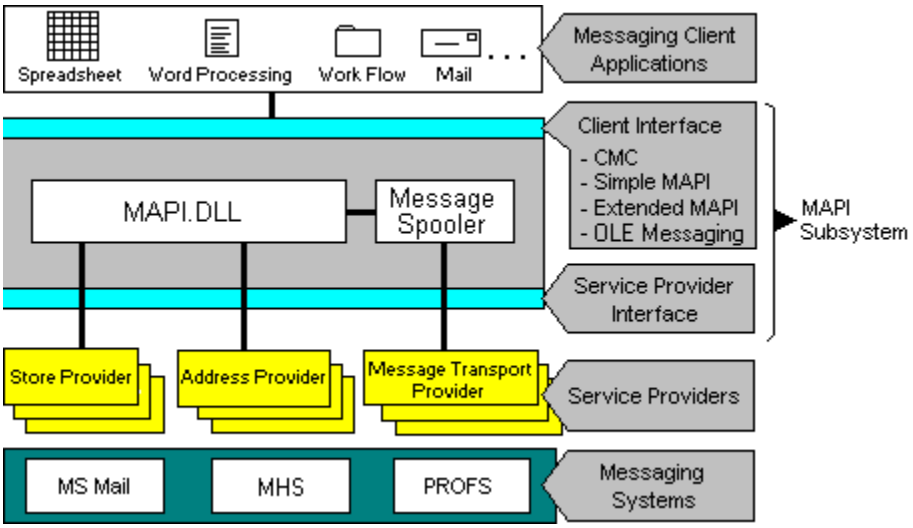


Figure 1. The MAPI architecture.

To use the messaging services, a client must first establish a *session*. A session is a specific connection between the client and the MAPI interface based on information provided in a *profile*. The profile contains configuration and user preference information. For example, the profile contains the names of various supporting files, the time interval to check for new messages, and other settings, such as whether to remember the user's password or to prompt the user for the password during each logon. A successful logon is required to enable the client's use of the MAPI system.

After establishing a MAPI session, the client can use the MAPI services. MAPI defines three primary services: Address Books, Message Transports, and Message Stores.

The *Address Book* service is similar to a telephone directory or Yellow Pages. The Address Book can be thought of as a permanent database that contains valid addressing information. An entry in the Address Book is called an *address entry* and consists of a display name, e-mail type, and e-mail address. The display name refers to the name, such as a person's full name, that an application displays to its users. You can provide a display name, and the Address Book service looks up the display name and provides the corresponding messaging system address.

The *Message Transport* supports communication between different devices and different underlying messaging systems.

The *Message Store* stores messages in a hierarchical structure that consists of one or more *folders*. A folder can be a *personal folder* that contains an individual's messages, or a *public folder*, similar to a bulletin board or online forum, which is accessible to many users. Each folder can contain *messages* or

other folders. A message represents a communication that is sent from the sender to one or more recipients or that gets posted in a public folder. A message can include an *attachment*, a document that is attached to and sent with the message.

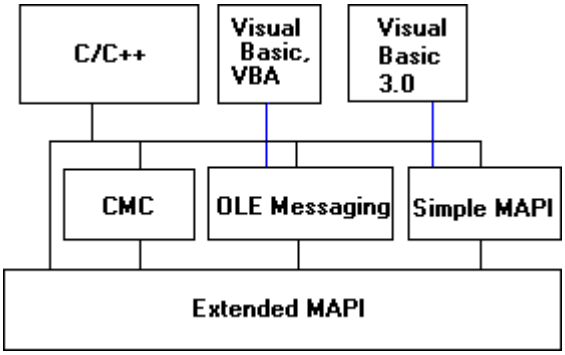
Several properties can be associated with the message: its subject, importance, and delivery properties, such as the time it is sent and received, and whether to notify the sender when the message is delivered and read. Some message properties identify the message as part of a *conversation*. The conversation properties allow you to group related messages and identify the sequence of comments and replies in the thread of the conversation.

The message can have one or more *recipients;* a recipient can be an individual or a *distribution list.* The distribution list can contain individuals and other distribution lists. For messages that are posted to public folders, the recipient can also be the public folder itself. Before sending a message, you can *resolve* each recipient; this means you should check each recipient against the Address Book to make sure the messaging address is valid.

MAPI 1.0 Programming Interfaces

Microsoft provides several programming interfaces for MAPI 1.0, so that developers working in a wide variety of development environments can use this common message exchange.

The following figure shows the OLE Messaging interface as a layer that is built on top of Extended MAPI 1.0. This is similar to the way that function calls to the Common Messaging Calls (CMC) interface are mapped to the underlying Extended MAPI interfaces. It also demonstrates that the OLE Messaging interface is available to both Visual Basic and C/C++ programmers:



It is important to recognize that the OLE Messaging Library does not offer access to all of the features of Extended MAPI. In particular, it is designed primarily for clients and is not suitable for service providers.

The following table summarizes the programming interfaces provided by Microsoft for MAPI version 1.0.

Programming interface	**Description**
MAPI custom controls	User interface elements for Visual Basic 3.0 developers. (Note: These will be superseded by the OLE Messaging Library version 1.0.)
Simple MAPI	API functions for C/C++ client developers that allow access to the Inbox (no access to MAPI properties). Most developers should probably use either CMC or Extended MAPI rather than Simple MAPI.
OLE Messaging Library version 1.0	**Programmable messaging objects for Visual Basic/VBA and C/C++ developers**
Common Messaging	API functions for C/C++ client developers; X.400

Calls (CMC)	API Association (XAPIA) standard.
Extended MAPI	OLE interfaces for C/C++ developers. Full access to all MAPI programming interfaces. Implemented by service providers and called by clients.

The following sections contrast differences between the OLE Messaging object and other programming interfaces.

Comparing MAPI Custom Controls and OLE Messaging

Although both the MAPI custom controls and the OLE Messaging Library are designed for Visual Basic programmers, they represent significantly different capabilities.

A *control* is a user interface element that enables you to display data for the user. The custom controls are usually more convenient to use or offer more specialized capabilities than the standard user interface controls, such as the list box, combo box, command button, and option button.

A programmable object may offer some user interface capabilities, but that is usually not its primary purpose. It offers the very powerful ability to interact with existing OLE objects. For a familiar example, consider the data access objects provided with Microsoft Visual Basic version 3.0 Professional Edition and subsequent versions. The data access library lets you create and use such database objects as tables, dynasets, and queries. As the data access library lets you use database objects, the OLE Messaging Library lets you add messaging to your applications.

The existing MAPI controls for use with Simple MAPI and Visual Basic 3.0 will be superseded by the OLE Messaging Library.

Comparing MAPI API and OLE Messaging

Compared to the function-call interfaces of traditional API libraries, an OLE Automation object library yields faster development and code that is easier to read, debug and maintain.

The Messaging object library also takes care of many programming details for you, such as memory management and keeping count of the number of objects in collections.

The following table compares a traditional function-call interface, such as CMC or Simple MAPI, with the OLE Messaging object interface.

Task or code	**Function-call interface**	**OLE Messaging**
Dim mFiles() As MapiFile Dim mRecips() As MapiRecip	Requires arrays of these structures to be declared, even if the developer does not use them.	Automatically manages these structures as child objects of the parent Message object.
ReDim mRecips(0) ReDim mFiles(0)	Structures are resized by re-dimensioning arrays.	Objects are added to collections with the **Add** method.
mMessage.RecipCount = 1	Requires developer to indicate the number of recipients and attachments.	Automatically determines the number of objects in these collections.
Error handling	Each function call returns an error code.	Integrated with Visual Basic error handling during both design and run-time.
Return values	Returned implicitly in the parameters of the	Returned as an explicit result of a method or in

function call. object properties.

As programming tasks grow more complex, the function-call approach becomes increasingly unwieldly. In contrast, the OLE Messaging Library expands gracefully to encompass greater complexity. A well-planned, thorough framework of collections, objects, methods, and properties can neatly encompass very complex systems.

A Short Tour of OLE Automation

The OLE Messaging Library is based on the capabilities provided by OLE Automation. The OLE Messaging Library allows you to create instances of programmable messaging *objects* that you can reference with tools that support OLE Automation, such as Visual Basic.

For the purposes of this documentation, an *object* is an OLE Automation object; a software component that exposes its properties and methods. Such an object follows the Visual Basic programming model and lets you get properties, set properties, and call methods.

You can think of programmable objects as additions or extensions to the programmable objects that are offered as part of Visual Basic, such as forms and controls. Forms and controls expose their properties and methods so that developers can tailor these objects for the needs of their program. In addition to the forms and controls, Visual Basic allows for the definition of a wide variety of other programmable objects by providing the **CreateObject** and **LoadObject** functions. Note that these functions do not have specialized names, like "CreateSpreadsheet" or "CreateDatabase." They are general-purpose functions that enable an open-ended number of programmable objects, including the OLE Messaging Library.

Throughout this topic, Visual Basic will be used as a concrete example of a tool that supports OLE Automation, but the statements about Visual Basic apply to all such tools.

Visual Basic scripts drive the OLE Messaging Library. The scripts can also drive other libraries that support OLE Automation, such as the libraries of programmable objects provided by Microsoft Excel 5.0 and Microsoft Access 2.0. Visual Basic can call many different programmable object libraries and can act as the glue that holds all of these objects together.

Each library can create its own objects, set properties, and call methods. The Visual Basic program coordinates the work of all the libraries; for example, it can direct the Microsoft Access object to find data in a specific table, direct the Microsoft Excel object to run calculations using that data, and then direct OLE Messaging Library objects to create a message that contains the results of those calculations and send the message to several recipients.

OLE Messaging Object Design

The OLE Messaging Library is designed for ease of use and convenience. It implements the Extended MAPI functions most used by client applications. The OLE Messaging Library is not designed for development of service providers. (For more information about service providers, see "A Short Tour of Microsoft MAPI 1.0."

This section of the documentation describes the design of the OLE Messaging Library.

Note This OLE Messaging Library design does not represent a one-to-one mapping to Extended MAPI objects. The description of the OLE Messaging object design does not always apply to the Extended MAPI programming interface.

The OLE Messaging Library version 1.0 defines the following objects: AddressEntry, Attachment, the Attachments collection, Field, the Fields collection, Folder, the Folders collection, Message, the Messages collection, Recipient, the Recipients collection, and Session.

The objects supported in the OLE Messaging Library version 1.0 can be grouped into three categories:

- *High-level objects*
- *Child objects* that are created automatically when the high-level objects are created
- *Collections*, or groups of objects of the same type

The following topics offer an overview of these categories.

High-Level Objects

The high-level objects include the Session, Folder, and Message objects. Other objects are accessible only from these high-level objects.

C programmers can access all high-level objects. Visual Basic programmers can use the Visual Basic **CreateObject** function with the string "MAPI.Session" to create the Session object.

In your Visual Basic application, you must usually use code of the following form to create the high-level session object:

```
Dim objSession As Object
Set objSession = CreateObject("MAPI.Session")
```

The following table describes the string that C programmers should use for each high-level object used by the **CreateObject** or **LoadObject** function:

High-level OLE Messaging object	String for CreateObject or LoadObject
Folder	"MAPI.Folder"
Message	"MAPI.Message"
Session	"MAPI.Session"

High-Level Objects and Child Objects

All OLE Messaging objects can be considered as relative to a Session object. The following diagram shows the logical hierarchy for the OLE Messaging Library:

```
Session
     Folder
           Messages Collection
                 Message
                       Recipients Collection
                             Recipient
                                   AddressEntry
                       Attachments Collection
                             Attachment
                       Fields Collection
                             Field
           Folders Collection
                 Folder...
```

In addition to the hierarchy of objects, each object has properties and methods. But the hierarchy is important because it determines the correct syntax to use in your Visual Basic applications. In your Visual Basic code, the relationship between a parent object and a child object is denoted by the left-to-right sequence of the objects in the Visual Basic statement.

OLE Messaging Object Collections

A *collection* is a group of objects of the same type. In the OLE Messaging Library, the name of the collection takes the plural form of the individual OLE Messaging object. For example, the "Messages" collection is the name of the collection that contains "Message" objects. The OLE Messaging Library supports the following collections: Attachments, Fields, Folders, Messages, and Recipients.

You can think of two kinds of collections: small collections and large collections.

For small collections, the OLE Messaging Library maintains a count of the number of objects in the collection. The Attachments, Recipients, and Fields collections can be characterized this way. You can add and delete items from the collection, and access individual items using an index into the collection.

Small collections, with a known number of member objects, have the property **Item**, the property **Count,** and an implied temporary property **Index,** assigned by the OLE Messaging Library. **Index** properties are valid only during the current MAPI session and can change as your application adds and deletes objects. The first **Index** value is 1.

For example, in an attachments collection with three attachments, the first attachment is referred to as Attachments.Item(1), the second as Attachments.Item(2), and the third as Attachments.Item(3). If your application deletes the second attachment, the third attachment becomes the second and Attachments.Item(3) has the value **Nothing**. The **Count** property is always equal to the highest **Index** in the collection.

Other applications may add and delete objects while your application is running. The **Count** property is not updated until you recreate or refresh the collection. For example, you call the Message object's **Update** method to refresh the count in its Attachments and Recipients collections.

For large collections, the OLE Messaging Library does not maintain a count of the number of objects. The Messages and Folders collections can be characterized as large collections. Instead of keeping a count, the collections support methods that let you get the first, next, previous, and last item in the collection.

For large collections, with an unknown number of member objects, MAPI assigns a permanent, unique string **ID** property when the individual member object is created. These IDs do not change from one MAPI session to another. You can call the Session object's **GetFolder** or **GetMessage** methods, specifying the unique ID, to obtain the individual folder or message objects. You can also use the **GetFirst** and **GetNext** methods to move from one object to the next in these collections.

Note When you want to use a collection, create a variable that refers to that collection to ensure

correct operation of the **GetFirst, GetNext, GetPrevious,** and **GetLast** methods.

For example, the following two code fragments are not equivalent:

```
' sample 1:  the collection returns the same message both times!
Set objMessage = objInBox.Messages.GetFirst
...
Set objMessage = objInBox.Messages.GetNext

' sample 2:  use an explicit variable to refer to the collection;
'   the Get methods return two different messages
Set objMsgColl = objSession.Inbox.Messages
Set objMessage = objMsgColl.GetFirst
...
Set objMessage = objMsgColl.GetNext
```

Code sample 1 causes the OLE Messaging Library to create a new Messages collection and to reinitialize the value of the collection's "current message." The **GetFirst** and **GetNext** method calls return the same value for objMessage.

Code sample 2 uses the existing collection objMsgColl, so the **GetFirst** and **GetNext** calls behave as expected for collections with more than one item.

The collections in the OLE Messaging Library are specifically designed for messaging applications. The definition of collections in this document may differ slightly from the definitions of collections in the OLE programming documentation. Where there are differences, the description of the operation of the OLE Messaging Library supersedes the other documentation.

Programming Tasks

This section describes some of the common programming tasks that you can perform with the OLE Messaging Library.

Category	Programming task described in this chapter
General Programming Tasks	Handling Errors Improving Application Performance Starting a Session with MAPI Viewing MAPI Properties
Working with Messages	Adding Attachments to a Message Customizing a Folder or Message Checking for New Mail Creating and Sending a Message Deleting a Message Making Sure the Message Gets There Reading a Message from the Inbox Searching for a Message Securing Messages
Working with Addresses	Changing an Existing Address Entry Selecting Recipients from the Address Book Using Addresses
Working with Folders	Accessing Folders Copying a Message to Another Folder Customizing a Folder or Message Moving a Message to Another Folder Searching for a Folder
Working with Public Folders	Posting Messages to a Public Folder Working with Conversations

Note that you cannot create new distribution lists, new folders, or new address book entries using version 1.0 of the OLE Messaging Library. However, you can use other applications or tools, such as the Microsoft Exchange client, to create these objects. After you create the objects, you can then access them using the OLE Messaging Library.

The following table summarizes the programming procedures that you must use to perform these tasks. Note that all tasks require a Session object and successful logon.

Programming task	Procedure
Accessing Folders	1. Access the Folder object's **Folders** property to obtain its collection of subfolders. 2. Use the Folders collection's **GetFirst**, **GetNext**, **GetPrevious**, and **GetLast** methods to traverse through the subfolders.
Adding Attachments to a Message	1. Create or obtain the Message object that is to include the attachment. 2. Call the Message object's Attachments collection's **Add** method**.**
Changing an Existing Address Entry	1. Obtain a valid AddressEntry object. 2. Update the **Name**, **Type**, or **Address**

	properties. 3. Call the **Update** method.
Checking for New Mail	Maintain a count of the number of messages in the Inbox folder that have the **Unread** property set to **True.** - or - Sort messages by time and count messages received after a specified time.
Copying a Message to Another Folder	1. Obtain the source message that you want to copy. 2. Call the destination folder's Messages collection's **Add** method, supplying the source message properties as parameters. 3. Copy the source Message object's **Sender** and **Recipients** properties to the new Message object. 4. Call the new Message object's **Update** method.
Creating and Sending a Message	1. Call the Messages collection's **Add** method to create a Message object. 2. Set the Message object's **Text**, **Subject**, and other message properties. 3. Call the message's Recipients collection's **Add** method to add a recipient. 4. Set the Recipient object's **Name, Address, or AddressEntry** property. 5. Call the Recipient object's **Resolve** method to validate the address information. 6. Call the Message object's **Send** method.
Customizing a Folder or Message	1. Create or obtain the Folder or Message object that will have the custom properties. 2. Call the object's Fields collection's **Add** method**.**
Deleting a Message	1. Select the message you want to delete. 2. Call the Message object's **Delete** method.
Handling Errors	Use the Microsoft Visual Basic "On Error Goto" construct to add exception-handling code just as you would in any Visual Basic application.
Improving Application Performance	Each dot in a Visual Basic statement directs the OLE Messaging Library to create a temporary internal object. Use explicit variables when you reuse messaging objects.

Making Sure the Message Gets There	1. Set the Message object's **DeliveryReceipt** and/or **ReadReceipt** property to **True.** 2. Call the Message object's **Send** method.
Moving a Message to Another Folder	Use the same procedure as "Copying A Message To Another Folder," and then delete the original source message from its folder.
Posting Messages to a Public Folder	1. Use a similar procedure as "Creating and Sending a Message," where you specify the name of the public folder as the Recipient name. Or 1. Call the public folder's Messages collection's **Add** method to create a Message object. 2. Set the Message object's **Text**, **Subject**, **ConversationSubject, ConversationIndex, TimeSent, TimeReceived,** and other message properties. 3. Set the Message object's **Unread**, **Submitted, and Sent** properties to True. 4. Call the Message object's **Send** or **Update** method to post the message.
Reading a Message from the Inbox	1. Call the Session's Inbox folder's **GetFirst, GetNext, GetPrevious,** and **GetLast** methods to obtain a Message object. 2. Obtain the Message object's **Text** property.
Searching for a Folder	Use the Session object's **GetFolder** method to obtain the folder from its known ID value. - or - Call the Folders collection's **Get*** methods to get individual folder objects. You can then compare properties of each folder with the desired properties.
Searching for a Message	Use the Session object's **GetMessage** method to obtain the message from its known ID value. - or - Call the Messages collection's **Get*** methods to get individual message objects. You can then compare properties of each message with the desired properties.
Securing Messages	1. Set the Message object's **Encrypted**

	and/or **Signed** properties to **True**. 2. Perform processing on the message's **Text** property to encrypt or sign the message . 3. Call the Message object's **Send** method.
Selecting Recipients from the Address Book	1. Call the Session's **AddressBook** method to use the MAPI AddressBook dialog. 2. Set a Recipients collection object to the Recipients collection returned by the **AddressBook** dialog. 3. Use that Recipients collection or copy individual recipients from it.
Starting a Session with MAPI	1. Create or obtain a Session object. 2. Call the Session object's **Logon** method.
Using Addresses	1. Set the message's Recipient object's **Address** property to a full address. 2. Call the Recipient object's **Resolve** method.
Viewing MAPI Properties	Specify the **Fields** item with a MAPI property tag.
Working with Conversations	1. Set the message's **ConversationTopic** property. 2. Set the message's **ConversationIndex** property. 3. Send the message by calling the **Send** method.- or - 3. Post the message in the public folder by setting the **Submitted** property to **True**.

The following sections describe these tasks in detail.

The following sections often discuss the hierarchy of the OLE Messaging objects. It is important to understand the hierarchy, because the hierarchical relationships between objects determine the correct syntax of Visual Basic statements. The relative positions of these objects in the hierarchy indicate how the objects appear from left to right in a Visual Basic statement.

In the sample code that appears in this documentation, individual statements are often broken across several lines. The underscore character "_" appears as a line continuation character, indicating that the statement is continued on the next line. This construct is used in an attempt to make the material easy to read.

All sample code that appears in this documentation is also available in the form of an Excel 5.0 spreadsheet that contains VBA modules. For information about the spreadsheet that contains the sample code, see the Release Notes for the MAPI 1.0 PDK.

See Also

Programmer's Reference

Accessing Folders

Folders can be organized in a hierarchy, allowing you to access folders within folders. A child folder within a parent folder is also called a *subfolder*. Subfolders appear within the parent folder object's Folders collection.

You cannot use the OLE Messaging Library version 1.0 to create new folders. However, after another application, such as the Microsoft Exchange client, has created a folder, you can use the OLE Messaging Library to access the folder.

There are two general approaches for accessing folders:

- Obtaining the folder directly by calling the Session **GetFolder** method
- Traversing folders using the Folders collection **Get*** methods

To obtain the folder directly using the **GetFolder** method, you must have the folder's identifier. In the following example, the identifier is stored in the variable *strFolderID*.

```
Function Session_GetFolder()
    On Error GoTo error_olemsg

    If objSession Is Nothing Then
        MsgBox "No active session, must log on"
        Exit Function
    End If
    If strFolderID = "" Then
        MsgBox ("Must first set folder ID variable; see Folder->ID")
        Exit Function
    End If
    Set objFolder = objSession.GetFolder(strFolderID)
    'equivalent to:
    ' Set objFolder = objSession.GetFolder(folderID:=strFolderID)
    If objFolder Is Nothing Then
        Set objMessages = Nothing
        MsgBox "Unable to retrieve folder with specified ID"
        Exit Function
    End If
    MsgBox "Folder set to " & objFolder.Name
    Set objMessages = objFolder.Messages
    Exit Function

error_olemsg:
    MsgBox "Error " & Str(Err) & ": " & Error$(Err)
    Set objFolder = Nothing
    Set objMessages = Nothing
    MsgBox "Folder is no longer available; no active folder"
    Exit Function
End Function
```

To traverse through the hierarchy of folders, start with a known or available folder, such as the Inbox or Outbox, and examine its Folders collection. You can use the Folders collection's **GetFirst** and **GetNext** methods to get each folder in the collection. When you have a subfolder, you can examine its properties, such as its name, to see whether it is the desired folder. The following sample code traverses through all existing subfolders of the Inbox:

```
Function TestDrv_Util_ListFolders()
    On Error GoTo error_olemsg
    If objFolder Is Nothing Then
```

```
        MsgBox "must select a folder object; see Session menu"
        Exit Function
    End If
    If 2 = objFolder.Class Then  ' verify Folder object
        x = Util_ListFolders(objFolder)  ' use current global folder
    End If
    Exit Function

error_olemsg:
    MsgBox "Error " & Str(Err) & ": " & Error$(Err)
    Resume Next

End Function

' Function: Util_ListFolders
' Purpose: Recursively list all folders below the current folder
' See documentation topic: Folders collection
Function Util_ListFolders(objParentFolder As Object)
Dim objFoldersColl As Object ' the child Folders collection
Dim objOneSubfolder As Object 'a single Folder object
    On Error GoTo error_olemsg
    If Not objParentFolder Is Nothing Then
        MsgBox ("Folder name = " & objParentFolder.Name)
        Set objFoldersColl = objParentFolder.Folders
        If Not objFoldersColl Is Nothing Then ' loop through all
            Set objOneSubfolder = objFoldersColl.GetFirst
            While Not objOneSubfolder Is Nothing
                x = Util_ListFolders(objOneSubfolder)
                Set objOneSubfolder = objFoldersColl.GetNext
            Wend
        End If
    End If
    Exit Function

error_olemsg:
    MsgBox "Error " & Str(Err) & ": " & Error$(Err)
    Resume Next
End Function
```

See Also

Searching for a Folder

Adding Attachments to a Message

You can add one or more attachments to a message. You add each attachment to the Attachments collection, using the Message object's **Attachments** property. The relationship between the Message object and an attachment is shown as follows:

Message object
  Attachments collection
    Attachment object
      Type property
      Source property

The OLE Messaging Library supports three different kinds of attachments: files, links to files, and OLE objects. The attachment type is specified by its **Type** property. To add an attachment, use the related Attachment object properties or methods appropriate for that type, as shown in the table below:

Attachment type	**Related attachment object property or method**
mapiFileData	**ReadFromFile** method
mapiFileLink	**Source** property
mapiOle	**ReadFromFile** method

The following example demonstrates inserting a file as an attachment with the **ReadFromFile** method. This example assumes that the application has already created the Session object variable **objSession** and successfully called the Session object's **Logon** method, as described in the section, "Starting a Session with MAPI."

```
' Function: Attachments_Add_Data
' Purpose: Demonstrate the Add method for type = mapiFileData
' See documentation topic: Adding Attachments To A Message,
'    Add method (Attachments collection)
Function Attachments_Add_Data()
Dim objMessage As Object  ' local
Dim objRecip As Object    ' local

    On Error GoTo error_olemsg
    If objSession Is Nothing Then
        MsgBox ("must first log on; use Session->Logon")
        Exit Function
    End If
    Set objMessage = objSession.Outbox.Messages.Add
    If objMessage Is Nothing Then
        MsgBox "could not create a new message in the Outbox"
        Exit Function
    End If
    With objMessage  ' message object
        .Subject = "attachment test"
        .Text = "Have a nice day."
        Set objAttach = .Attachments.Add         ' add an attachment
        If objAttach Is Nothing Then
            MsgBox "Unable to create new Attachment object"
            Exit Function
        End If
        With objAttach
            .Type = mapiFileData
        .Position = 0  ' Exchange viewer displays at end of message
```

```
        .Source = "c:\smiley.bmp"
         End With
    .Update   ' update the message
    End With
    MsgBox "Created message, added 1 mapiFileData attachment, updated"
    Exit Function

error_olemsg:
    MsgBox "Error " & Str(Err) & ": " & Error$(Err)
    Resume Next

End Function
```

Note that setting a position value within the message can cause some viewers, such as the Microsoft Exchange client, to overwrite the character that appears at that position in the message. You can insert the attachment at various places in the message text:

```
' objMessage and objAttach as defined above
    objMessage.Text = " " & objMessage.Text ' add space for attachment
    objAttach.position = 1
    objMessage.Update
```

The OLE Messaging Library does not actually place the attachment within the message; that is the responsibility of the messaging client application. However, to avoid these display problems with some viewers, you can specify a position value that contains a negative value, zero, or a large positive value.

For example, given a a negative position value, the Microsoft Exchange client inserts the attachment before the message without overwriting any characters of the message. Given a positive value greater than the number of characters in the message, the Microsoft Exchange client displays the attachment after the message. A value of zero also causes the client to display the attachment after the message.

To insert an attachment of type **mapiOLE**, use code similar to the **mapiFileData** type example. Set the attachment type to **mapiOLE** and make sure that the specified file is a valid OLE docfile (a file saved by an OLE-aware application such as Microsoft Word 6.0 that uses the OLE interfaces **IStorage** and **IStream**).

To add an attachment of type **mapiFileLink**, set the **Type** property to **mapiFileLink** and set the **Source** property to the file name. The following sample code demonstrates this type of attachment:

```
' Function: Attachments_Add
' Purpose: Demonstrate the Add method for type = mapiFileLink
' See documentation topic: Adding Attachments To A Message,
'     Add method (Attachments collection)
Function Attachments_Add()
    On Error GoTo error_olemsg

    If objAttachColl Is Nothing Then
        MsgBox "must first select an attachments collection"
        Exit Function
    End If
    Set objAttach = objAttachColl.Add         ' add an attachment
    With objAttach
        .Type = mapiFileLink
        .Position = 0   ' place at end of message
        .Source = "\\server\bitmaps\honey.bmp"  ' modify UNC name
    End With
    ' must update the message to save the new info
```

```
    objOneMsg.Update   ' update the message
    MsgBox "Added an attachment of type mapiFileLink"
    Exit Function

error_olemsg:
    MsgBox "Error " & Str(Err) & ": " & Error$(Err)
    Resume Next

End Function
```

See Also

Position Property (Attachment Object)

Creating and Sending a Message

Changing an Existing Address Entry

OLE Messaging version 1.0 lets you change existing address entries in the Personal Address Book. Use the following procedure to change existing address entries:

1. Select the AddressEntry object to modify. You can obtain the AddressEntry object in several ways, including the following:

 Call the Session object's **AddressBook** method to let the user select recipients. The method returns a Recipients collection. Examine each recipient object's **AddressEntry** property to obtain its child AddressEntry object.

 Use the Message object's **Sender** property to obtain an AddressEntry object.

 Examine a Message object's Recipients collection to obtain an individual recipient object, then use its **AddressEntry** property to obtain its child AddressEntry object.

2. Change individual properties of the AddressEntry object, such as the **Name**, **Address**, or **Type** properties.
3. Call the AddressEntry object's **Update** method.

Note that OLE Messaging only supports changes to the Personal Address Book.

The following sample code demonstrates this procedure:

```
' Function: AddressEntry_Update
' Purpose: Demonstrate the Update method
'     (Note: OLE Messaging v1.0 only affects the PAB)
' See documentation topic: Update method AddressEntry object
Function AddressEntry_Update()
Dim objRecipColl As Object    ' Recipients collection
Dim objNewRecip As Object     ' New recipient

    On Error GoTo error_olemsg
    If objSession Is Nothing Then
        MsgBox "must log on first"
        Exit Function
    End If
    Set objRecipColl = objSession.AddressBook  ' let user select
    If objRecipColl Is Nothing Then
        MsgBox "must select someone from the address book"
        Exit Function
    End If
    Set objNewRecip = objRecipColl.Item(1)
    With objNewRecip.AddressEntry
        .Name = .Name & " the Magnificent"
        .Type = "X.500"  ' you can update the type, too...
        .Update
    End With
    MsgBox "Updated an address entry name: " & _
            objNewRecip.AddressEntry.Name
    Exit Function

error_olemsg:
    MsgBox "Error " & Str(Err) & ": " & Error$(Err)
    Resume Next

End Function
```

See Also

Using Addresses

Selecting Recipients from the Address Book

Checking for New Mail

The Inbox contains new messages. When users refer to new messages, they can indicate messages that arrive after the last time that they read messages, or they can indicate all unread messages. Depending on the needs of your application users, your applications can check various message properties to determine whether there is new mail.

The following sample code tracks new messages by checking for messages in the Inbox with the **Unread** property value **True**.

```
' Function: Util_CountUnread
' Purpose:  Count unread messages in a folder
' See documentation topic: Checking For New Mail;
'    Unread property (Message)
Function Util_CountUnread()
Dim cUnread As Integer         ' counter

    On Error GoTo error_olemsg
    If objMessages Is Nothing Then
        MsgBox "must select a messages collection"
        Exit Function
    End If
    Set objMessage = objMessages.GetFirst
    cUnread = 0
    While Not objMessage Is Nothing ' loop through all messages
        If True = objMessage.Unread Then
            cUnread = cUnread + 1
        End If
        Set objMessage = objMessages.GetNext
    Wend
    MsgBox "Number of unread messages = " & cUnread
    Exit Function

error_olemsg:
    MsgBox "Error " & Str(Err) & ": " & Error$(Err)
    Resume Next

End Function
```

You can also check for new messages by counting the messages received after a specified time. For example, your application can maintain a variable that represents the time of the latest message received, based on the message object's **TimeReceived** property. The application can periodically check for all messages with a **TimeReceived** value greater than the saved value. When new messages are found, the application updates its count of new messages and updates the saved value.

See Also

TimeReceived Property (Message Object)

Reading a Message from the Inbox

Copying a Message to Another Folder

The procedure documented in this section demonstrates a way to copy message properties using the Message object **Add** method that is supported in OLE Messaging version 1.0.

To copy a message from one folder to another folder using OLE Messaging version 1.0, use the following procedure:

1. Obtain the source message that you want to copy.
2. Call the destination folder's Messages collection's **Add** method, supplying the source message properties as parameters.

The hierarchy of objects is as follows:

Session object
  Folder object
    Messages collection
      Message object

To obtain the source message that you want to copy, first obtain its folder, then obtain the message within the folder. For more information about finding messages, see the section, "Searching for a Message."

To obtain the destination folder, you can use the following approaches:


Use the Folders collection's **Get*** methods to search for a specific folder.

Call the Session object's **GetFolder** method with a string parameter that specifies the FolderID, a unique identifier for that folder.

For more information about finding folders, see the section, "Searching for a Folder."

The following example demonstrates how to copy the first message that appears in the Inbox folder. The message is copied to the Outbox, but could as easily be copied to any folder with a known identifier and therefore accessible using the Session object's **GetFolder** method. This example assumes that the application has already created the Session object variable **objSession** and successfully called the Session object's **Logon** method, as described in the section, "Starting a Session with MAPI."

```
'/*******************************/
' Function: Util_CopyMessage
' Purpose: Utility functions that demonstrates code to copy a message
' See documentation topic: Copying A Message To Another Folder
Function Util_CopyMessage()
' obtain the source message to copy
' for this sample, just use the first message in the Inbox
' assume session object already created
Dim objDestFolder As Object   ' destination folder
Dim objCopyMsg As Object      ' new message that is the copy
Dim strRecipName As String    ' copy of recipient name from original message
Dim i As Integer              ' loop counter

    On Error GoTo error_olemsg
    If objOneMsg Is Nothing Then
        MsgBox "must first select message"
        Exit Function
    End If
    If objFolder Is Nothing Then
        MsgBox "must first select a folder"
        Exit Function
    End If
```

```
    strFolderID = objFolder.Id
    ' Copy to the destination folder
    Set objDestFolder = objSession.GetFolder(strFolderID)
    If objDestFolder Is Nothing Then
        MsgBox "Unable to create destination folder for ID " _
                & strFolderID
        Exit Function
    Else
        MsgBox "Copying message to destination folder " _
                & objDestFolder.Name
    End If
    Set objCopyMsg = objDestFolder.Messages.Add _
       (Subject:=objOneMsg.Subject, _
        Text:=objOneMsg.Text, _
        Type:=objOneMsg.Type, _
        importance:=objOneMsg.importance)
    If objCopyMsg Is Nothing Then
        MsgBox "Unable to create new message in destination folder"
        Exit Function
    End If
    ' copy all the recipients
    For i = 1 To objOneMsg.Recipients.Count Step 1
        strRecipName = objOneMsg.Recipients.Item(i).Name
        If strRecipName <> "" Then
            Set objOneRecip = objCopyMsg.Recipients.Add
            If objOneRecip Is Nothing Then
                MsgBox "unable to create recipient in message copy"
                Exit Function
            End If
            objOneRecip.Name = strRecipName
        End If
    Next i
    ' copy other properties; a few listed here as an example
    objCopyMsg.Sent = objOneMsg.Sent
    objCopyMsg.Text = objOneMsg.Text
    objCopyMsg.Unread = objOneMsg.Unread
    objCopyMsg.Update
    ' if *moving* a message to another folder, delete the original msg:
    '     objOneMsg.Delete
    ' move operation implies that the original message is removed
    Exit Function

error_olemsg:
    MsgBox "Error " & Str(Err) & ": " & Error$(Err)
    Exit Function    ' so many steps to succeed; just exit on error

End Function
```

Note that this procedure does not preserve *all* message properties.

See Also

Moving a Message to Another Folder

Creating and Sending a Message

Creating and sending a message is easy when you use the OLE Messaging Library. Use the following procedure:

1. Establish a session with the MAPI system.
2. Call the Messages collection's **Add** method to create a Message object.
3. Supply values for the Message object's **Subject** and **Text** properties.
4. Call the Recipients collection's **Add** method for each recipient.
5. Call the Message object's **Send** method.

The following sample demonstrates each of these steps for a message sent to a single recipient:

```
' This sample also appears as the "Quick Start" sample in Chapter 1
Function QuickStart()
Dim objSession As Object    ' Session object
Dim objMessage As Object    ' Message object
Dim objOneRecip As Object   ' Recipient object

    On Error GoTo error_olemsg

    ' create a session then log on, supplying username and password
    Set objSession = CreateObject("MAPI.Session")
    ' change the parameters to valid values for your configuration
    objSession.Logon 'profileName:="Princess Leia", _
                'profilePassword:="go_rebels"

    ' create a message and fill in its properties
    Set objMessage = objSession.Outbox.Messages.Add
    objMessage.Subject = "Gift of droids"
    objMessage.Text = "Help us, Obi-wan. You are our only hope."

    ' create the recipient
    Set objOneRecip = objMessage.Recipients.Add
    objOneRecip.Name = "Obi-wan Kenobi"
    objOneRecip.Type = mapiTo
    objOneRecip.Resolve

    ' send the message and log off
    objMessage.Update
    objMessage.Send showDialog:=False
    MsgBox "The message has been sent"
    objSession.Logoff
    Exit Function

error_olemsg:
    MsgBox "Error " & Str(Err) & ": " & Error$(Err)
    Resume Next

End Function
```

Note When you edit objects other than the Message object, save your changes using the **Update** method before you clear or reuse the object variable. If you do not use the **Update** method, your changes can be lost without warning.

After calling the message object's **Send** method, you should not try to access the Message object

again. The **Send** method invalidates the Message object.

See Also

Adding Attachments to a Message

Customizing a Folder or Message

Customizing a Folder or Message

The OLE Messaging Library allows customization and extensibility by offering the Field object and Fields collection. A field object includes a name, a data type, and a value property. An object that supports fields, in effect, lets you add your own custom properties to the object.

The OLE Messaging Library version 1.0 supports the use of fields with the Message and Folder objects.

For example, consider that you want to add a "Keyword" property to messages so that you can associate a string with the message. You may wish to use a self-imposed convention that values of the "Keyword" are restricted to a small set of strings. You can then organize your messages by the "Keyword" property.

The following example shows how to add the field to the Message object.

```
' Function: Fields_Add
' Purpose:  Add a new field object to the Fields collection
' See documentation topic:  Add method (Fields collection)
Function Fields_Add()
Dim cFields As Integer      ' count of Fields in the collection
Dim objNewField As Object   ' new Field object

    On Error GoTo error_olemsg
    If objFieldsColl Is Nothing Then
        MsgBox "must first select Fields collection"
        Exit Function
    End If
    Set objNewField = objFieldsColl.Add( _
                      Name:="Keyword", _
                      Class:=vbString, _
                      Value:="Peru")
    If objNewField Is Nothing Then
        MsgBox "could not create new Field object"
        Exit Function
    End If
    cFields = objFieldsColl.Count
    MsgBox "new Fields collection count = " & cFields
    ' you can now write code that searches for
    ' messages with this "custom property"
    Exit Function

error_olemsg:
    MsgBox "Error " & Str(Err) & ": " & Error$(Err)
    Resume Next

End Function
```

Note that the new field information specified by the **Add** method is not actually saved until you call the Message object's **Update** method.

For a complete list of the valid Field object data types, see the reference documentation for the Fields collection **Add** method.

See Also

Field Object

Deleting a Message

The Message object's **Delete** method deletes the message. Use the following procedure:

1. Select the message you want to delete.
2. Call the message object's **Delete** method.
3. Set the message object to **Nothing**.

You should not try to access the Message object after deleting it.

See Also

Searching for a Message

Handling Errors

The OLE Messaging Library version 1.0 raises exceptions for all errors. When you write Visual Basic applications that use the OLE Messaging Library, use the same error handling techniques that you use in all your Visual Basic applications: the Visual Basic "On Error Goto" construct.

Note that the error-handling techniques vary slightly using Visual Basic 3.0 and Visual Basic for Applications, as supported in Microsoft Excel 5.0. For more information, see your product's Visual Basic documentation.

The OLE Messaging Library collects error information from different *levels* of software. The lowest level of software is that which interacts directly with hardware, such as a mouse driver or video driver. Higher levels of software move toward greater device independence and greater generality.

The following diagram suggests the different levels of software in Visual Basic applications that use the OLE Messaging Library. Visual Basic applications reside at the highest level and interact with the MAPI OLE Messaging Library at the next lower level. The MAPI OLE Messaging Library interacts with the MAPI system software, and the MAPI system software interacts with a lower layer of software, the operating system.

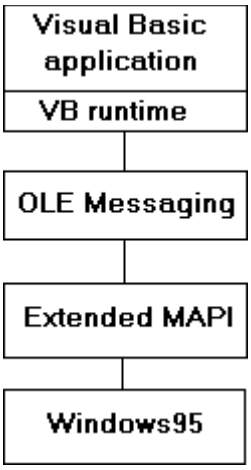


Figure 1. Software components interact with others at lower and higher levels. The Visual Basic application at the top of this diagram interacts with the VB run-time library, which interacts with the OLE Messaging Library; the OLE Messaging Library interacts with Extended MAPI; and Extended MAPI in turn interacts with the operating system, here shown as Windows95.

Errors can occur at any level or at the interface between any two levels. For example, a user of your application without security permissions can be denied access to an address book entry. The lowest level in this diagram, the operating system, returns the error to the next higher level, and so on, until the error is returned to the highest level, the Visual Basic application.

The OLE Messaging Library reports the error to Visual Basic so that Visual Basic raises an runtime exception. You can then handle the exception in the same way that you handle other error exceptions.

```
' demonstrates error handling for Logon
' Function: TestDrv_Util_CreateSessionAndLogon
' Purpose: Call the utility function Util_CreateSessionAndLogon
' See documentation topic:  Handling Errors;
'    Creating And Sending A Message
Function TestDrv_Util_CreateSessionAndLogon()
Dim bFlag As Boolean
    On Error GoTo error_olemsg
    bFlag = Util_CreateSessionAndLogon()
    MsgBox "bFlag = " & bFlag
```

```
    Exit Function

error_olemsg:
    MsgBox "Error " & Str(Err) & ": " & Error$(Err)
    Resume Next

End Function

' Function: Util_CreateSessionAndLogon
' Purpose: Demonstrate common error handling for Logon
' See documentation topic: Handling Errors
Function Util_CreateSessionAndLogon() As Boolean
    On Error GoTo err_CreateSessionAndLogon

    Set objSession = CreateObject("MAPI.Session")
    objSession.Logon
    Util_CreateSessionAndLogon = True
    Exit Function

err_CreateSessionAndLogon:
    If (Err = 1275) Then  ' VB4.0 uses "Err.Number"
        MsgBox "User pressed Cancel"
    Else
        MsgBox "Unrecoverable Error:" & Err
    End If
    Util_CreateSessionAndLogon = False
    Exit Function

error_olemsg:
    MsgBox "Error " & Str(Err) & ": " & Error$(Err)
    Resume Next

End Function
```

Note that this sample uses the **Err** object and is tailored for Visual Basic for Applications. When you use Visual Basic 3.0, use the **Err** function to obtain the status code and the **Error$** function to obtain a descriptive error message, as in the following example:

```
' Visual Basic 3.0 error handling
MsgBox "Error number " & Err & " description: " & Error$(Err)
```

Depending on your version of Microsoft Visual Basic, the error code will be returned as a long integer or as a short integer, and you should appropriately define the value of the error codes checked by your program, such as the value 1275 in the sample above.

The example checks for an error corresponding to the Extended MAPI error code MAPI_E_USER_CANCEL. The short integer error code represents the value 1000 + the low-order 16 bits of the MAPI error code, which in this case is 275. So a Visual Basic application that uses short integer codes should check for the value 1275.

The long integer error code is displayed as the negative value -2147221229 or as the hexadecimal value &h80040113. The value in the low-order 16 bits represents the MAPI error code. In this example &h0113, or the decimal value 275, indicates MAPI_E_USER_CANCEL.

You can obtain errors from many different layers of the software. For example, the hexadecimal return value 80070057 indicates MAPI_E_INVALID_PARAMETER, an error code that is defined as part of the Win32® SDK. This error is returned by Microsoft Windows NT™ or Microsoft Windows95. For a

comprehensive list of error codes, see the Microsoft MAPI SDK documentation and your operating system SDK documentation.

The values of many Extended MAPI error codes appear in the header file MAPICODE.H, which is provided with the Microsoft MAPI 1.0 SDK.

See Also

OperatingSystem Property (Session Object)

Starting a Session with MAPI

Improving Application Performance

This section describes how your Visual Basic code can operate most efficiently when you use messaging objects. Note that this section is written primarily for Visual Basic programmers rather than for C programmers.

To access OLE Messaging objects, you create Visual Basic statements that concatenate the object names in sequence from left to right, separating objects with a "dot," the period character. For example, consider the following Visual Basic statement:

```
Set objMessage = objSession.Inbox.Messages.GetFirst
```

The OLE Messaging Library creates an internal object for each dot that appears in the statement. For example, the portion of the statement that says "objSession.Inbox" directs OLE Messaging to create an internal Folder object that represents the user's Inbox. The next portion, ".Messages," directs OLE Messaging to create an internal Messages collection object. The final part, ".GetFirst," directs OLE Messaging to create an internal Message object that represents the first message in the user's Inbox. The statement contains three dots; the OLE Messaging Library creates three internal objects.

The best rule of thumb is to remember that dots are expensive. For example, the following two lines of code are very inefficient:

```
' warning: do not code this way, this is inefficient
MsgBox "Text: " & objSession.Inbox.Messages.MoveFirst.Text
MsgBox "Subj: " & objSession.Inbox.Messages.MoveFirst.Subject
```

While this code generates correct results, it is not very efficient. For the first statement, the OLE Messaging Library creates internal objects that represent the Inbox, its Messages collection, and its first message. After the application displays the text, these internal objects are discarded. In the next line, the same internal objects are generated again. A more efficient approach would be to generate the internal objects only once:

```
With objSession.Inbox.Messages.MoveFirst
  MsgBox "Text: " & .Text
  MsgBox "Subj: " & .Subject
End With
```

When your application needs to use an object more than once, define a variable for the object and set its value. The following sample code is very efficient when your application reuses the Folder, Messages collection, or Message objects:

```
' very efficient when the objects will be reused
Set objInboxFolder = objSession.Inbox
Set objInMessages = objInboxFolder.Messages
Set objOneMessage = objInMessages.MoveFirst
With objOneMessage
  MsgBox "The Message Text: " & .Text
  MsgBox "The Message Subject: " & .Subject
End With
```

Now that you understand that a dot in a statement directs the OLE Messaging Library to create a new internal object, it is easy to see that the following sample is not correct:

```
' error: collection returns the same message both times
MsgBox("first msg: " & inBoxObj.Messages.GetFirst)
MsgBox("next msg: " & inBoxObj.Messages.GetNext)
```

The OLE Messaging Library creates a temporary internal object that represents the Messages collection, then discards it after displaying the first message. The second statement directs OLE Messaging to create another new temporary object that represents the Messages collection. This Messages collection is new and has no state information; that is, this new collection has not called **GetFirst**. The **GetNext** statement causes it to display its first message again.

Use the Visual Basic **With** statement or explicit variables to generate the expected results. The following example uses explicit variables:

```
' Use the Visual Basic With statement
With objSession.Inbox.Messages
    Set objMessage = .GetFirst
    '...
    Set objMessage = .GetNext
End With
' Use explicit variables to refer to the collection;
Set objMsgColl = objSession.Inbox.Messages
Set objMessage = myMsgColl.GetFirst
...
Set objMessage = myMsgColl.GetNext
```

For more information about improving the performance of your applications, see your Microsoft Visual Basic programming documentation.

See Also

Handling Errors

Making Sure the Message Gets There

The Message object contains two properties that can direct the underlying MAPI system to report successful receipt of the message: **DeliveryReceipt** and **ReadReceipt**.

When you set these properties to **True** and send the message, the underlying MAPI system automatically tracks the message for you. When you set the **DeliveryReceipt** property, the MAPI system automatically generates a message to the sender reporting when the recipient receives the message. When you set the **ReadReceipt** property, the MAPI system automatically generates a message to the sender reporting when the recipient reads the message.

See Also

Securing Messages

Moving a Message to Another Folder

The procedure documented in this section demonstrates a way to move message properties using the Message object **Add** and **Delete** methods supported in OLE Messaging version 1.0. To move a message from one folder to another folder using OLE Messaging version 1.0, use the following procedure:

1. Obtain the source message that you want to copy.
2. Call the destination folder's Messages collection's **Add** method, supplying source message properties as parameters.
3. Call the source message's **Delete** method to delete the original source message from its folder.

For the complete sample, see the section, "Copying a Message to Another Folder." The final lines of code for the procedure should delete the original message:

```
' "Move" implies explicit delete of the initial message
objOneMsg.Delete
```

See Also

Copying a Message to Another Folder

Posting Messages to a Public Folder

You can post messages to a public folder using two different methods:

- Use the **Send** method to send the message to the public folder, as you would send the message to an individual.
- Use the public folder's Messages collection's **Add** method to create the message within the public folder. When you are ready to make the message available, call the **Send** or **Update** method.

Regardless of the approach you use, you must set a few more message properties than you would when sending a message to a recipient. When you post a message to a public folder, the components of the MAPI architecture that usually handle a message and set its properties do not manage the message. Your application must set the **Unread**, **Submitted**, and **Sent** properties to **True**; and must set the **TimeSent** and **TimeReceived** properties to the current time.

Note When posting messages in a public folder, you cannot use OLE Messaging Library version 1.0 to set the **Sender** property. These Sender and related underlying properties are not present for a message created by the OLE Messaging Library.

When you use the Message object's **Send** method to post the message, you use the same procedure that you use to create and send a message to an individual, with one minor exception: you specify *the name of the public folder* as the message recipient. Public folders are available in the address book, so you can use the Session object's **AddressBook** method and the Recipient object's **Resolve** method with public folder names. For more information about the complete procedure for sending messages, see "Creating And Sending A Message."

The remainder of this section describes the procedure for creating a message within the public folder itself.

1. Call the Messages collection's **Add** method to create a Message object.
2. Set the Message object's **Text**, **Subject**, **ConversationSubject, ConversationIndex, TimeSent, TimeReceived,** and other message properties as desired.
3. Set the Message object's **Unread, Submitted,** and **Sent** properties to **True**.
4. Call the Message object's **Send** or **Update** method.

Note that when you post a message, you must explicitly set the **TimeSent** and **TimeReceived** properties. When you send a message using the **Send** method, the MAPI system assigns the values of these properties for you. However, when you post the message by setting the **Submitted** property, your application must set the time properties. Set both time properties to the same value, just before you set the **Submitted** property to **True**.

```
' Function: Util_New_Conversation
' Purpose: Set properties to start a new conversation in a public folder
' See documentation topic: Working With Conversations;
'     Posting Messages To A Public Folder
Function Util_NewConversation()
Dim objRecipColl As Object
Dim i As Integer
Dim objNewMsg As Object      ' new message object
Dim strNewIndex As String
    On Error GoTo error_olemsg

    If objSession Is Nothing Then
        MsgBox "No session object - Use Session->Logon"
        Exit Function
    End If
    Set objNewMsg = objSession.Outbox.Messages.Add
    If objNewMsg Is Nothing Then
```

```
        MsgBox "unable to create a new message for the public folder"
        Exit Function
    End If
    strConversationFirstMsgID = objNewMsg.Id   'save for reply
    With objNewMsg
        .Subject = "used space vehicle wanted"
        .ConversationTopic = .Subject
        .ConversationIndex = Util_GetEightByteTimeStamp() ' utility
        .Text = "Wanted: Apollo or Mercury spacecraft with low mileage."
           ' or you could pick the public folder from the address book
        Set objOneRecip = .Recipients.Add(Name:="Car Ads", Type:=mapiTo)
        If objOneRecip Is Nothing Then
            MsgBox "Unable to create the public folder recipient"
            Exit Function
        End If
        .Recipients.Resolve
        ' when you create the msg in a public folder, set properties:
        '.TimeSent = Time
        '.TimeReceived = .TimeSent
        '.Submitted = True
        '.Unread = True
        '.Sent = True
        .Update
        .Send showDialog:=False
    End With
    Exit Function

error_olemsg:
    MsgBox "Error " & Str(Err) & ": " & Error$(Err)
    Resume Next

End Function
```

For complete information about the **ConversationIndex** property, see "Working With Conversations."

See Also

Searching For a Folder

Creating and Sending a Message

Working with Conversations

Reading a Message from the Inbox

After establishing a Session object and successfully logging on to the system, the user can access her *Inbox*. The Inbox is the default folder for mail received by the user.

As described in the section titled "OLE Messaging Object Design," the OLE Messaging objects are organized in a hierarchy. The Session object at the topmost level allows access to a Folder. Each Folder contains a Messages collection, and the Messages collection contains individual Message objects. The text of the message appears in the Message object's **Text** property:

Session object
  Folder object
    Messages collection
      Message object
        Text property

To obtain an individual message, the application must move down through this object hierarchy to the **Text** property. The following example uses the Session object's **Inbox** property to obtain a Folder object, then uses the Folder object's **Messages** property to obtain a Messages collection object, and calls the Messages collection's methods to get a specific message.

This example assumes that the application has already created the Session object variable **objSession** and successfully called the Session object's **Logon** method, as described in the section, "Starting a Session with MAPI."

```
Dim objSession As Object      ' Session object
Dim objInboxFolder As Object  ' Folder object
Dim objInMessages As Object   ' Messages collection
Dim objOneMsg As Object       ' Message object
...
' move down through the hierarchy
Set objInboxFolder = objSession.Inbox
Set objInMessages = objInboxFolder.Messages
Set objOneMsg = objInMessages.GetFirst
MsgBox "The message text: " & objOneMsg.Text
```

Note Use the Visual Basic keyword **Set** whenever you initialize a variable that represents an object. When you set an object variable without using the **Set** keyword, Visual Basic generates an error message.

The example above declares several object variables. However, it is also possible to access the message with fewer variables. The following sample is equivalent to the sample code above:

```
Set objOneMsg = objSession.Inbox.Messages.GetFirst
MsgBox "The message text: " & objOneMsg.Text
```

You should declare an individual variable when the application needs to access an object more than once. When an object is accessed repeatedly, variables can help make your code efficient. For more information, see the section, "Improving Application Performance."

See Also

Creating and Sending a Message

Improving Application Performance

Searching for a Message

Searching for a Folder

Two frequently used folders, the Inbox and the Outbox, are available through Session object properties. To access these folders, simply set a Folder object to the corresponding property.

To access other folders, search for the folder using one of the following techniques:

- Call the Session object's **GetFolder** method with a string parameter that specifies the FolderID, a unique identifier for the folder.
- Use the **Get*** methods to traverse through the Folders collection. Search for a specific folder by comparing the current folder's properties with the desired properties.

Each approach is described in detail in the following sections.

Using the Session Object GetFolder Method

When you know the unique identifier for the folder you are looking for, you can call the Session object's **GetFolder** method.

The unique identifier for the folder, established at the time the folder is created, is stored in its **ID** property. The ID is a string representation of the MAPI entry ID and its value is determined by the service provider.

The following code fragment contains code that saves the ID for the folder, then uses it in a subsequent **GetFolder** call:

```
' Function: Session_GetFolder
' Purpose: Demonstrate how to set a folder object
' See documentation topic: Session object GetFolder method
Function Session_GetFolder()
    On Error GoTo error_olemsg

    If objSession Is Nothing Then
        MsgBox "No active session, must log on"
        Exit Function
    End If
    If strFolderID = "" Then
        MsgBox ("Must first set folder ID variable; see Folder->ID")
        Exit Function
    End If
    Set objFolder = objSession.GetFolder(strFolderID)
    'equivalent to:
    ' Set objFolder = objSession.GetFolder(folderID:=strFolderID)
    If objFolder Is Nothing Then
        Set objMessages = Nothing
        MsgBox "Unable to retrieve folder with specified ID"
        Exit Function
    End If
    MsgBox "Folder set to " & objFolder.Name
    Set objMessages = objFolder.Messages
    Exit Function

error_olemsg:
    MsgBox "Error " & Str(Err) & ": " & Error$(Err)
    Set objFolder = Nothing
    Set objMessages = Nothing
    MsgBox "Folder is no longer available; no active folder"
    Exit Function
```

```
End Function
```

Using the Get Methods

When you are looking for a folder within a Folders collection, you can traverse down through the collection, examining properties of each folder object to determine whether it is the folder you want.

OLE Messaging Library version 1.0 supports the **GetFirst**, **GetLast**, **GetNext**, and **GetPrevious** methods for the Folders collection object.

The following sample demonstrates how to use the **Get*** methods to search for the specified folder.

```
' Function: TestDrv_Util_GetFolderByName
' Purpose: Call the utility function Util_GetFolderByName
' See documentation topic: Item property (Folder object)
Function TestDrv_Util_GetFolderByName()
Dim fFound As Boolean
    fFound = Util_GetFolderByName("Junk mail")
    If fFound Then
        MsgBox "Folder named 'Junk mail' found"
    Else
        MsgBox "Folder named 'Junk mail' not found"
    End If
    Exit Function

error_olemsg:
    MsgBox "Error " & Str(Err) & ": " & Error$(Err)
    Resume Next
End Function

' Function: Util_GetFolderByName
' Purpose: Use Get* methods to search for a folder
' See documentation topic:  Searching For a Folder
Function Util_GetFolderByName(strSearchName As String) As Boolean
Dim objOneFolder As Object   ' local; temp version of folder object

    On Error GoTo error_olemsg
    Util_GetFolderByName = False  ' default; assume failure
    If objFolder Is Nothing Then
        MsgBox "must first select a folder; such as Session->Inbox"
        Exit Function
    End If
    Set objFoldersColl = objFolder.Folders  ' Folders collection
    If objFoldersColl Is Nothing Then
        MsgBox "no subfolders; not found"
        Exit Function
    End If
    ' get the first folder in the collection
    Set objOneFolder = objFoldersColl.GetFirst
    ' loop through all the folders in the collection
    Do While Not objOneFolder Is Nothing
        If objOneFolder.Name = strSearchName Then
            Exit Do  ' found it, leave the loop
        Else  ' keep searching
            Set objOneFolder = objFoldersColl.GetNext
        End If
```

```
    Loop
    ' exit from the do while loop comes here
    ' if objOneFolder is valid, the folder is found
    If Not objOneFolder Is Nothing Then
        Util_GetFolderByName = True    ' success; set to False above
    End If
    Exit Function

error_olemsg:
    MsgBox "Error " & Str(Err) & ": " & Error$(Err)
    Resume Next
End Function
```

You can also navigate upward by using the **Parent** property.

See Also

Searching for a Message

Searching for a Message

To access a message, search for the message using one of the following techniques:

- Call the Session object's **GetMessage** method with a string parameter that specifies the MessageID, a unique identifier for the message.
- Use the **Get*** methods to traverse through the folder's Messages collection. Search for a specific message by comparing the current message object's properties with the desired properties.

Each approach is described in detail in the following sections.

Using the Session Object GetMessage Method

When you know the unique identifier for the message you are looking for, you can call the Session object's **GetMessage** method.

The message ID specifies a unique identifier that is created for the message object at the time it is created. The ID is accessible through the object's **ID** property.

The following code fragment contains code that saves the ID for the folder, then uses it in a subsequent **GetMessage** call:

```
' Function: Session_GetMessage
' Purpose: Demonstrate how to set a message object using GetMessage
' See documentation topic: GetMessage method (Session object)
Function Session_GetMessage()
    On Error GoTo error_olemsg

    If objSession Is Nothing Then
        MsgBox "No active session, must log on"
        Exit Function
    End If
    If strMessageID = "" Then
        MsgBox ("Must first set message ID variable; see Message->ID")
        Exit Function
    End If
    Set objOneMsg = objSession.GetMessage(strMessageID)
    If objOneMsg Is Nothing Then
        MsgBox "Unable to retrieve message with specified ID"
        Exit Function
    End If
    MsgBox "GetMessage returned msg with subject: " & objOneMsg.Subject
    Exit Function

error_olemsg:
    MsgBox "Error " & Str(Err) & ": " & Error$(Err)
    Set objOneMsg = Nothing
    MsgBox "Message is no longer available; no active message"
    Exit Function
End Function
```

Using the Get Methods

When you are looking for a message within a Messages collection, you can traverse through the collection, examining properties of each message object to determine if it is the message you want.

OLE Messaging Library version 1.0 supports the **GetFirst**, **GetLast**, **GetNext**, and **GetPrevious** methods for the Messages collection object.

The following sample demonstrates how to use the **Get*** methods to search for the specified message.

```
' Function: TestDrv_Util_GetMessageByName
' Purpose: Call the utility function Util_GetMessageByName
' See documentation topic: Item property (Message object)
Function TestDrv_Util_GetMessageByName()
Dim fFound As Boolean
    On Error GoTo error_olemsg

    fFound = Util_GetMessageByName("Junk mail")
    If fFound Then
        MsgBox "Message named 'Junk mail' found"
    Else
        MsgBox "Message named 'Junk mail' not found"
    End If
    Exit Function

error_olemsg:
    MsgBox "Error " & Str(Err) & ": " & Error$(Err)
    Resume Next
End Function

' Function: Util_GetMessageByName
' Purpose: Use Get* methods to search for a message
' See documentation topic:  Searching for a message
' search through the messages for one with a specific subject
Function Util_GetMessageByName(strSearchName As String) As Boolean
Dim objOneMessage As Object   ' local; temp version of message object

    On Error GoTo error_olemsg
    Util_GetMessageByName = False  ' default; assume failure
    If objFolder Is Nothing Then
        MsgBox "must first select a folder; such as Session->Inbox"
        Exit Function
    End If
    Set objMessages = objFolder.Messages
    Set objOneMessage = objMessages.GetFirst
    If objOneMessage Is Nothing Then
        MsgBox "no messages in the folder"
        Exit Function
    End If
    ' loop through all the messages in the collection
    Do While Not objOneMessage Is Nothing
        If objOneMessage.Subject = strSearchName Then
            Exit Do  ' found it, leave the loop
        Else  ' keep searching
            Set objOneMessage = objMessages.GetNext
        End If
    Loop
    ' exit from the do while loop comes here
    ' if objOneMessage is valid, the message was found
    If Not objOneMessage Is Nothing Then
        Util_GetMessageByName = True    ' success
    End If
    Exit Function
```

```
error_olemsg:
    MsgBox "Error " & Str(Err) & ": " & Error$(Err)
    Resume Next

End Function
```

See Also

Searching for a Message

Securing Messages

The Message object contains two properties that specify security for the message: the **Encrypted** and **Signed** properties. When you want to request that your message be secured, set these flags to **True.**

Note that these flags simply represent a *request* to the underlying messaging service. Whether the message is encrypted or signed depends on whether these security measures are implemented by your messaging service.

Neither MAPI nor OLE Messaging Library version 1.0 performs encryption or digital signing. The OLE Messaging Library simply sets the appropriate Extended MAPI properties so that the proper request for security is delivered to the messaging service. For more information about the capabilities of your messaging service, contact your Microsoft Exchange system administrator.

```
Dim objMessage As Object   ' assume valid Message object
'...
objMessage.Encrypted = True
objMessage.Send
```

See Also

Making Sure the Message Gets There

Selecting Recipients From the Address Book

After establishing a Session object and successfully logging onto the system, the user can access the address book to select recipients. You can select recipients from any address book, such as the Global Address List or the Personal Address Book.

As described in "OLE Messaging Object Design," the OLE Messaging objects are organized in a hierarchy. The Session object at the topmost level contains an **AddressBook** method that lets your application users select recipients from an address book. The method returns a Recipients collection, which contains individual Recipient objects. The Recipient object in turn specifies an AddressEntry object. This hierarchy is shown in the following diagram:

Recipients collection
  Recipient object
    Address property (full address)
    AddressEntry object
      Address property (e-mail address)
      Type property

To obtain an individual **Address** property that can be used to address and send messages, the application must move down through this object hierarchy. The following code example uses the Recipients collection returned by the Session object's AddressBook method.

This example assumes that the application has already created the Session object variable **objSession** and successfully called the Session object's **Logon** method, as described in the section, "Starting a Session with MAPI."

```
' Function: Session_AddressBook
' Purpose: Set the global variable that contains the current recipients
'    collection to that returned by the Session AddressBook method
' See documentation topic: AddressBook method (Session object)
Function Session_AddressBook()
    On Error GoTo err_Session_AddressBook

    If objSession Is Nothing Then
        MsgBox "must first create MAPI session and logon"
        Exit Function
    End If
    Set objRecipColl = objSession.AddressBook( _
        Title:="Select Attendees", _
        forceResolution:=True, _
        recipLists:=1, _
        toLabel:="&OLE Messaging")  ' appears on button
    ' Note: initial value not used in version 1.0
    ' parameter not used in call: Recipients:=objInitRecipColl
    MsgBox "Name of first recipient = " & objRecipColl.Item(1).Name
    Exit Function

err_Session_AddressBook:
    If (Err = 91) Then   ' MAPI dlg-related function that sets an object
        MsgBox "No recipients selected"
    Else
        MsgBox "Unrecoverable Error:" & Err
    End If
    Exit Function
End Function
```

See Also

Changing an Existing Address Entry

Using Addresses

Starting a Session with MAPI

As described in the section "OLE Messaging Object Design," all messaging objects are relative to the Session object. One of the first tasks of every application is to create a valid Session object and call its **Logon** method.

The Session object is created using the Visual Basic function **CreateObject**. The following code demonstrates how to perform this common startup task:

```
Function Util_CreateSessionAndLogon() As Boolean
    On Error GoTo err_CreateSessionAndLogon

    Set objSession = CreateObject("MAPI.Session")
    objSession.Logon
    Util_CreateSessionAndLogon = True
    Exit Function

err_CreateSessionAndLogon:
    If (Err = 1275) Then  ' VB4.0 uses "Err.Number"
        MsgBox "User pressed Cancel"
    Else
        MsgBox "Unrecoverable Error:" & Err
    End If
    Util_CreateSessionAndLogon = False
    Exit Function

End Function
```

When no parameters are supplied to the **Logon** method, as in the example above, the OLE Messaging Library displays an application-modal logon dialog box that prompts the application user to select a user profile. Based on the characteristics of the selected profile, the underlying MAPI system logs on the user or prompts for password information.

You can also choose to use your own application's dialog box to obtain the parameters needed to log on, rather than using the MAPI logon dialog box. The following example obtains the profile name and password information and directs the **Logon** method not to display a logon dialog box:

```
' Function: Session_Logon_NoDialog
' Purpose: Call the Logon method, set parameter to show no dialog
' See documentation topic: Logon Method (Session object)
Function Session_Logon_NoDialog()
    On Error GoTo error_olemsg
    ' can set strProfileName, strPassword from a custom form
    ' adjust these parameters for your configuration
    If objSession Is Nothing Then
        Set objSession = CreateObject("MAPI.Session")
    End If
    If Not objSession Is Nothing Then
    ' configure these parameters for your needs either here
    ' or in the function Util_Initialize
        objSession.Logon profileName:=strProfileName, _
                         showDialog:=False
    End If
    Exit Function

error_olemsg:
    If 1273 = Err Then
```

```
        MsgBox "cannot logon: incorrect profile name or password"
        Exit Function
    End If
    MsgBox "Error " & Str(Err) & ": " & Error$(Err)
    Resume Next
End Function
```

Note Your Visual Basic application should be able to handle cases that occur when a user provides incorrect profile or password information, or that occur when a user cancels from the Logon dialog box. For more information, see the section, "Handling Errors"

After establishing a Session object and successfully logging on to the system, the user has access to several default objects provided by the Session object, including the Inbox and Outbox folders. For more information, see the section, "Reading a Message from the Inbox."

See Also

Reading a Message from the Inbox

Creating and Sending a Message

Using Addresses

In general, Microsoft MAPI supports two kinds of addressing:


Addresses that the MAPI system looks up for you in your address book, based on a display name that you supply


Addresses that represent *custom addresses,* that are used as supplied without lookup

The OLE Messaging Library version 1.0 supports both kinds of addresses with its Recipient object. To look up an address name, you supply the **Name** property only. To use custom addresses, you supply the full address in the **Address** property.

This section describes the concepts involved in addressing, the design of the OLE Messaging Recipient and AddressEntry objects, and the procedures your applications can use when managing addresses.

The address book can be thought of as a database in persistent storage, managed by the MAPI system, that contains valid addressing information that is associated with a *display name.* The display name represents the way that a person's name might be displayed for your application users, using that person's full name, rather than the e-mail address that the messaging system uses to transmit the message. For example, the display name "John Doe" is mapped to the e-mail address "johnd".

In contrast with the address book, the objects that you create with the OLE Messaging Library are temporary objects that reside in memory. When you fill in the Recipient object's **Name** property with a display name, you must then *resolve* the address. To resolve the address means that you ask the MAPI system to look up the display name in the database and supply you with the corresponding address. When the display name is ambiguous, or can match more than one entry in the address book, the MAPI system prompts you to select from a list of possible matching names.

The Recipient **Name** property represents the display name. Call the Recipient object's **Resolve** method to resolve the display name.

After the Recipient object is resolved, it has a child AddressEntry object that contains a copy of the valid addressing information from the database. The child AddressEntry object is accessible from the Recipient's **AddressEntry** property. The Recipient object and AddressEntry object properties are related as follows:

OLE Messaging property	MAPI property	Description
Recipient.Address	Combination of PR_ADDR_TYPE and PR_EMAIL_ADDRESS	Full address; AddressEntry's **Type** and **Address** properties
Recipient.Name	PR_DISPLAY_NAME	Display name
Recipient.AddressEntry.Address	PR_EMAIL_ADDRESS	E-mail address
Recipient.AddressEntry.Type	PR_ADDR_TYPE	E-mail type
Recipient.AddressEntry.Name	PR_DISPLAY_NAME	Display name
Recipient.AddressEntry.ID	PR_ENTRYID	Unique identifier for the address entry

The Recipient **Address** property represents a *full address*, that is, the combination of address type and e-mail address that MAPI uses to send a message. The full address represents the same information that appears in the AddressEntry **Address** property and the AddressEntry **Type** property.

You can also supply a complete recipient address yourself. By manipulating the address yourself, you direct the MAPI system to send the message to the full address that you supply without using the

database. In this case, you must also supply the display name. When you supply a custom address, the Recipient **Address** property must use the following syntax:

TypeValue:*AddressValue*

There is also a third method of working with addresses: You can directly obtain and use the Recipient object's child AddressEntry object from messages that have already been successfully sent through the messaging system.

For example, to reply to a message, you can use a Message object's **Sender** property to get a valid AddressEntry object. When you work with valid AddressEntry objects, you do not have to call the **Resolve** method.

Note When you use existing AddressEntry objects, do not try to modify them. In general, do not write directly to the Recipient object's child AddressEntry object properties.

In summary, you can provide addressing information in three different ways:

- Obtain the correct addressing information for a known display name. Set the Recipient object's **Name** property and call the Recipient object's **Resolve** method. Note that the **Resolve** method can display a dialog.
- Use an existing valid address entry, such as the Message object's **Sender** property, when you are replying to a message. Set the Recipient object's **AddressEntry** property to an existing AddressEntry object that is known to be valid. (You do not need to call the **Resolve** method.)
- Create a custom address. Set the Recipient object's **Address** property, using the correct syntax as described above (use the colon character '**:**' to separate the address type from the address), and call the **Resolve** method.

The following sample code demonstrates these three kinds of addresses:

```
' Function: Util_UsingAddresses
' Purpose:  Set addresses three ways
' See documentation topic: Using Addresses
Function Util_UsingAddresses()
Dim objNewMessage As Object         ' new message object for example
Dim strAddrEntryID As String        ' ID value from AddressEntry object
Dim strName As String               ' Name from AddressEntry object
    On Error GoTo error_olemsg
    If objOneMsg Is Nothing Then
        MsgBox "Must select a message"
        Exit Function
    End If
    With objOneMsg.Recipients.Item(1).AddressEntry
        strAddrEntryID = .Id
        strName = .Name
    End With
    Set objNewMessage = objSession.Outbox.Messages.Add
    If objNewMessage Is Nothing Then
        MsgBox "Unable to add a new message"
        Exit Function
    End If
    ' add three recipients
    ' 1. look up entry in address book specified by profile
    Set objOneRecip = objNewMessage.Recipients.Add( _
        Name:=strName, _
        Type:=mapiTo)
    If objOneRecip Is Nothing Then
        MsgBox "Unable to add recipient using Display Name"
```

```
        Exit Function
    End If
    objOneRecip.Resolve
    ' 2. add a custom recipient
    Set objOneRecip = objNewMessage.Recipients.Add( _
        Address:="SMTP:davidhef@microsoft.com", _
        Type:=mapiTo)
    If objOneRecip Is Nothing Then
        MsgBox "Unable to add recipient using custom addressing"
        Exit Function
    End If
    objOneRecip.Resolve

    ' 3. add a valid address entry object, such as Message.Sender
    Set objOneRecip = objNewMessage.Recipients.Add( _
        entryID:=strAddrEntryID, _
        Name:=strName, _
        Type:=mapiTo)
    If objOneRecip Is Nothing Then
        MsgBox "Unable to add recipient using existing AddressEntry ID"
        Exit Function
    End If

    objNewMessage.Text = "expect 3 different recipients"
    MsgBox ("count = " & objNewMessage.Recipients.Count)
    ' you can also call resolve for the whole collection
    ' objNewMessage.Recipients.Resolve (True) ' resolve all; show dialog

    objNewMessage.Subject = "test"
    objNewMessage.Update           ' update the message
    x = objNewMessage.Send(showDialog:=False)
    Exit Function

error_olemsg:
    MsgBox "Error " & Str(Err) & ": " & Error$(Err)
    Exit Function

End Function
```

See Also

Sender property (Message Object)

Resolve method (Recipient Object)

Changing an Existing Address Entry

Viewing MAPI Properties

You can use a feature of the OLE Messaging Library's Fields collection object to view the values of MAPI properties.

The Fields collection **Item** property allows you to specify the actual *property tag* value as an identifier. A property tag is a 32-bit unsigned integer that contains the property identifier in its high-order 16 bits and the property type (its underlying data type) in the low-order 16 bits. Several macros are available to C/C++ programmers in the MAPI 1.0 SDK to help manipulate the property tag data structure. The macros PROP_TYPE and PROP_ID extract the property type and property identifer from the property tag. The macro PROP_TAG builds the property tag from the provided type and identifier components.

For complete reference information on MAPI properties and property tags, see the *MAPI Programmer's Reference*.

```
' Function: Fields_Selector
' Purpose: View a MAPI property by supplying a property tag value as
'    the Item value
' See documentation topics: Viewing MAPI Properties;
'    Item property (Fields collection)
Function Fields_Selector()
Dim lValue As Long
Dim strMsg As String

    On Error GoTo error_olemsg

    If objFieldsColl Is Nothing Then
        MsgBox "must first select a Fields collection"
        Exit Function
    End If
    ' you can provide a dialog here so users enter MAPI proptags...
    ' or select property names from a list; for now, hard-coded value
    lValue = &h1a001e
    ' &H1a = PR_MESSAGE_CLASS; &H001e = 30 = PT_STRING8
    ' high-order 16 bits is property id; low-order is property type
    Set objOneField = objFieldsColl.Item(lValue)
    If objOneField Is Nothing Then
        MsgBox "Could not get the Field using the value " & lValue
        Exit Function
    Else
        strMsg = "Used the value " & lValue & " to access the property"
        strMsg = strMsg & "PR_MESSAGE_CLASS: type = " & objOneField.Type
        strMsg = strMsg & "; value = " & objOneField.Value
        MsgBox strMsg
    End If
    Exit Function

error_olemsg:
    MsgBox "Error " & Str(Err) & ": " & Error$(Err)
    Resume Next

End Function
```

See Also

Field Object

Customizing a Folder or Message

Working With Conversations

Two Message object properties let you show complex relationships among messages by defining them as part of a *conversation*. A conversation is a series of messages, consisting of an initial message and all messages sent in reply to the initial message. When the initial message or a reply elicits additional messages, the resulting messages are called a *conversation thread*. A thread represents a subset of messages in the conversation.

The conversation properties **ConversationTopic** and **ConversationIndex** give you another way to organize and display messages. Rather than simply organizing messages by subject, time received, or sender, you can show rich and complex relationships among messages. The **ConversationTopic** property is a string that describes the overall subject of the conversation. The **ConversationIndex** property is an index that you can use to represent the relationships between messages and replies.

When you start an initial message, set the **ConversationTopic** property to an appropriate value that will apply to all messages within the conversation. For many applications, the message **Subject** property is appropriate.

You can use your own convention to decide how to use the **ConversationIndex** property. However, it is recommended that you adopt the same convention used by the Microsoft Exchange client message viewer, so that you can use that viewer's user interface to show the relationships between messages in a conversation.

By convention, Microsoft Exchange uses **ConversationIndex** values that represent concatenated time stamp values. The first time stamp in the string represents the original message. When a new message represents a reply to a conversation message, it copies the **ConversationIndex** string of the message it is replying to, and appends a time stamp value to the end of the string. The new string value is used as the **ConversationIndex** value of the new message.

When you use this convention, you can easily see relationships among messages when you sort the messages by **ConversationIndex** values.

The following code sample provides a utility function, **Util_GetEightByteTimeStamp**, which can be used to build Exchange-compatible **ConversationIndex** values. The utility function calls the OLE function **CoCreateGuid** to obtain the time stamp value from a GUID (globally unique identifier) data structure. The GUID value is composed of a time stamp and a machine identifier; the utility function saves the part that contains the time stamp.

```
' declarations for the Util_GetEightByteTimeStamp function
Type GUID
    Guid1 As Long
    Guid2 As Long
    Guid3 As Long
    Guid4 As Long
End Type
Declare Function CoCreateGuid Lib "COMPOBJ.DLL" (pGuid As GUID) As Long
' Note:  Use "OLE32.DLL" for Windows NT, Win95 platforms
Global Const S_OK = 0
' end declarations section

' Function: Util_GetEightByteTimeStamp
' Purpose: Generate a time stamp for use in conversations
' See documentation topic: Working With Conversations
Function Util_GetEightByteTimeStamp() As String
Dim lResult As Long
Dim lGuid As GUID
    ' Exchange conversation is a unique 8-byte value
    ' Exchange client viewer sorts by concatenated properties
```

```
    On Error GoTo error_olemsg

    lResult = CoCreateGuid(lGuid)
    If lResult = S_OK Then
        Util_GetEightByteTimeStamp = _
            Hex$(lGuid.Guid1) & Hex$(lGuid.Guid2)
    Else
        Util_GetEightByteTimeStamp = "00000000"   ' zeroes
    End If
    Exit Function

error_olemsg:
    MsgBox "Error " & Str(Err) & ": " & Error$(Err)
    Util_GetEightByteTimeStamp = "00000000"
    Exit Function

End Function
```

When you start a new conversation, set the **ConversationIndex** property to the value returned by this function, as follows:

```
' new conversation
objMessage.ConversationIndex = Util_GetEightByteTimeStamp()
```

When you are replying to a message in an existing conversation, append the time stamp value to that message's **ConversationIndex** value:

```
' reply within an existing conversation
Dim objOriginalMsg As Object    ' assume valid
Dim strNewIndex As String
'...
' copy the original topic and
' append the current time stamp to the original time stamp
objMessage.ConversationTopic = objOriginalMsg.ConversationTopic
strNewIndex = objOriginalMsg.ConversationIndex _
               & Util_GetEightByteTimeStamp()
objMessage.ConversationIndex = strNewIndex
```

For additional sample code, see "Posting Message To a Public Folder."

See Also

Posting Messages to a Public Folder

Programmer's Reference

This section contains property and method reference information for the OLE Messaging objects.

The following table summarizes each object's properties and methods:

Object	Properties	Methods
AddressEntry	Address, Application, Class, ID, Name, Parent, Session, Type	Delete, Details, Update
Attachment	Application, Class, Index, Name, Parent, Position, Session, Source, Type	Delete, ReadFromFile, WriteToFile
Attachments (collection)	Application, Class, Count, Item, Parent, Session	Add, Delete
Field	Application, Class, ID, Index, Name, Parent, Session, Type, Value	Delete, ReadFromFile, WriteToFile
Fields (collection)	Application, Class, Count, Item, Parent, Session	Add, Delete
Folder	Application, Class, Fields, FolderID, Folders, ID, MAPIOBJECT*, Messages, Name, Parent, Session, StoreID	(none)
Folders (collection)	Application, Class, Parent, Session	GetFirst, GetLast, GetNext, GetPrevious
Message	Application, Attachments, Class, ConversationIndex, ConversationTopic, DeliveryReceipt, Encrypted, Fields, FolderID, ID, Importance, MAPIOBJECT*, Parent, ReadReceipt, Recipients, Sender, Sent, Session, Signed, Size, StoreID, Subject, Submitted, Text, TimeReceived, TimeSent, Type, Unread	Delete, Options, Send, Update
Messages (collection)	Application, Class, Parent, Session	Add, Delete, GetFirst, GetLast, GetNext, GetPrevious, Sort
Recipient	Address, AddressEntry, Application, Class, Index, Name, Parent, Session, Type	Delete, Resolve
Recipients (collection)	Application, Class, Count, Item, Parent, Resolved, Session	Add, Delete, Resolve
Session	Application, Class, CurrentUser, Inbox, MAPIOBJECT*, Name, OperatingSystem, Outbox, Parent, Session, Version	AddressBook, GetAddressEntry, GetFolder, GetMessage, Logoff, Logon

*** The MAPIOBJECT property is not available to Visual Basic applications. For more information,**

please see the reference for the MAPIOBJECT property.

This section is organized by object, consisting of a brief summary of each object that lists its properties and methods, followed by reference documentation for the individual properties and methods. The properties and methods are organized alphabetically.

To avoid duplication, the initial section "All OLE Messaging Objects" describes the properties that have the same meaning for all OLE Messaging Library objects. This section includes the properties **Application, Class, Parent,** and **Session.**

See Also

A Short Tour

Programmer's Guide

All OLE Messaging Objects

All OLE Messaging Library objects contain the properties **Application, Class, Parent,** and **Session.** The **Application** and **Session** properties have the same values for all objects within a given session. The **Parent** property indicates the logical parent of the object. The **Class** property is an integer value that identifies the OLE Messaging object.

Note that for the Session object, the **Parent** and **Session** properties are assigned the value **Nothing**. The Session object represents the highest level in the OLE Messaging object hierarchy and has no parent.

To reduce duplication, the detailed reference for these common ("superclass") properties appears only once, in this section.

Many objects also have a **Type** property, but the **Type** property is not defined for all objects and its meaning varies depending on the object. The following table summarizes the **Type** property:

Object	Description of the Type property
AddressEntry	The messaging system; SMTP, Fax, etc.
Attachment	Attachment type; mapiFileData, mapiFileLink, or mapiOle
Field	The field data type; vbInteger, vbLong, etc.
Message	The message class; IPM.Note, etc.
Recipient	Recipient type; To:, Cc:, or Bcc: line

For detailed information about the **Type** property, see the reference documentation for each object.

The following table lists the properties that are common to all OLE Messaging objects and that have the same meaning for all objects:

Properties

Property name	Type	Access
Application	String	Read-only
Class	Long	Read-only
Parent	Object	Read-only
Session	Session object	Read-only

See Also

Programmer's Reference

Application Property (All OLE Messaging Objects)

Indicates the name of the active application, which is the OLE Messaging Library, "OLE Messaging." Read-only.

Syntax

object.**Application**

Data Type

String

Comments

The **Application** property always contains the string "OLE Messaging".

Note that this behavior for the Microsoft OLE Messaging Library differs from other implementations of OLE Automation servers. Many OLE Automation servers are based on executables (files that take the extension .EXE) and return an object value. The OLE Messaging Library is based on the MAPI subsystem, which is implemented by dynamic link libraries (files that take the extension .DLL).

Example

```
' Function: Session_Application
' Purpose: Display the Application property of the Session object
' See documentation topic: Application property
Function Session_Application()
Dim objSession As Object
    ' error handling
    Set objSession = CreateObject("MAPI.Session")
    If Not objSession Is Nothing Then
        MsgBox "Session's Application property = " _
            & objSession.Application
    End If
    ' error handling
End Function
```

See Also

Version Property (Session Object)

Class Property (All OLE Messaging Objects)

Indicates the OLE Messaging Library object. Read-only.

Syntax

object.**Class**

Data Type

Long

Comments

The **Class** property contains a numeric constant that identifies the OLE Messaging Library object. The following values are defined:

OLE Messaging Library object	Class ID value	Value
AddressEntry	8	mapiAddressEntry
Attachment	5	mapiAttachment
Attachments (collection)	21	mapiAttachments
Field	6	mapiField
Fields (collection)	22	mapiFields
Folder	2	mapiFolder
Folders (collection)	18	mapiFolders
Message	3	mapiMsg
Messages (collection)	19	mapiMessages
Recipient	4	mapiRecipient
Recipients (collection)	20	mapiRecipients
Session	0	mapiSession

Example

```
' Function: Util_DecodeObjectClass
' Purpose: Decode the long integer class value,
'          show the related object name
' See documentation topic: Class property (All OLE Messaging objects)
Function Util_DecodeObjectClass(lClass As Long)
    ' error handling here...
    Select Case (lClass)
        Case mapiSession:
            MsgBox ("Session object; class = " & lClass)
        Case mapiMsg:
            MsgBox ("Message object; class = " & lClass)
    End Select
    ' error handling
End Function

' Function: TestDrv_Util_DecodeObjectClass
```

```
' Purpose: Call the utility function DecodeObjectClass for Class values
' See documentation topic: Class property (All OLE Messaging objects)
Function TestDrv_Util_DecodeObjectClass()
    ' error handling here...
    If objSession Is Nothing Then
        MsgBox "Need to set the Session object: Session->Logon"
        Exit Function
    End If
    ' expect type mapiSession = 0 for Session object
    Util_DecodeObjectClass (objSession.Class)
    Set objMessages = objSession.Inbox.Messages
    Set objOneMsg = objSession.Inbox.Messages.GetFirst
    If objOneMsg Is Nothing Then  ' empty inbox
        Exit Function
    End If
    ' expect type mapiMessage = 3 for Message object
    Util_DecodeObjectClass (objOneMsg.Class)
    ' error handling here...
End Function
```

See Also

All OLE Messaging Objects

Parent Property (All OLE Messaging Objects)

Returns the parent of the object. Read-only.

Syntax

object.**Parent**

Data Type

Object

Comments

The **Parent** property in OLE Messaging Library version 1.0 currently returns the *immediate* parent of an object. For example, the immediate parent for each object is shown in the following table:

OLE Messaging object	Immediate Parent in object hierarchy
AddressEntry	Recipient object
Attachment	Attachments collection
Attachments (collection)	Message object
Field	Fields collection
Fields (collection)	Message or Folder object
Folder (Inbox, Outbox)	Session object
Folder (all others)	Folders collection
Folders (collection)	Folder object
Message	Messages collection
Messages (collection)	Folder object
Recipient	Recipients collection
Recipients (collection)	Message object
Session	(not defined)

Note that current plans call for the **Parent** property to be changed in subsequent releases. In future versions, the **Parent** property may return the *logical* parent of the object, not the *immediate* parent of the object. For example, a folder contains a messages collection, which contains message objects. In future releases, the **Parent** property for a message object would be set to its logical parent, the folder, rather than the immediate parent, the messages collection.

The Session object represents the highest level in the hierarchy of OLE Messaging objects and its **Parent** property is set to **Nothing**.

Example

This example displays the name of the parent messages collection of a message:

```
' Function: Message_Parent
Function Message_Parent()
    ' error handling here
    If objOneMsg Is Nothing Then
        MsgBox "Need to select a message; see Messages->Get*"
        Exit Function
    End If
    ' Immediate parent of message is the messages collection
```

```
    MsgBox "Message immediate parent class = " & objOneMsg.Parent.Class
    ' error handling code
End Function
```

To get to the folder, you have to take the parent of the Messages collection object::

```
' Function: Messages_Parent
' Purpose: Display the Messages collection Parent class value
' See documentation topic: Parent property (All OLE Messaging objects)
Function Messages_Parent()
    ' error handling here...
    If objMessages Is Nothing Then
        MsgBox "No active messages collection"
        Exit Function
    End If
    MsgBox "Messages collection parent has class value: " & _
            objMessages.Parent.Class
    Exit Function
    ' error handling here...
End Function
```

See Also

Class Property (All OLE Messaging Objects)

Folder Object

Message Object

Session Object

Session Property (All OLE Messaging Objects)

Returns the top-level Session object associated with the specified OLE Messaging Library object. Read-only.

Syntax

Set *objSession* = *object*.**Session**

Data Type

Object

Comments

The Session object represents the highest level in the OLE Messaging object hierarchy. Its **Session** property is set to **Nothing**.

Example

```
' Function: Folder_Session
' Purpose: Access the Folder's Session property and display its name
' See documentation topic: Session property (All OLE Messaging objects)
Function Folder_Session()
Dim objSession2 As Object   ' session object to get the property
    ' error handling here...
    If objFolder Is Nothing Then
        MsgBox "No active folder; please select Session->Inbox"
        Exit Function
    End If
    Set objSession2 = objFolder.Session
    If objSession2 Is Nothing Then
        MsgBox "Unable to access Session property"
        Exit Function
    End If
    MsgBox "Folder's Session property name = " & objSession2.Name
    Set objSession2 = Nothing
    ' error handling here...
End Function
```

See Also

Session Object

AddressEntry Object

The AddressEntry object defines valid addressing information for a given messaging system. An address usually represents a person or process to which the messaging system can deliver messages.

The AddressEntry object is often used as a child object of the Recipient object. In this context, the AddressEntry represents a copy of valid addressing information that is obtained from the Address Book during a call to the Recipient object's **Resolve** method. When you obtain the AddressEntry object in this context, you should not modify its properties.

Properties

Property name	Type	Access
Address	String	Read-write
Application	String	Read-only
Class	Long	Read-only
ID	String	Read-only
Name	String	Read-write
Parent	Object	Read-only
Session	Session object	Read-only
Type	String	Read-write

Methods

Method name	Parameters
Delete	(none)
Details	(optional) parentWindow as Long
Update	(none)

See Also

Recipient Object

Address Property (AddressEntry Object)

Specifies the messaging address of an address list entry or message recipient. Read-write.

Syntax

objAddrEntry.**Address**

Data Type

String

Comments

The AddressEntry object's **Address** property provides a unique string to identify a message recipient and routing information for messaging systems. The format of the address string is specific to each messaging system.

The AddressEntry object's **Address** and **Type** properties combine to form the *full address*, the complete messaging address that appears in the Recipient object's **Address** property. The Recipient object's **Address** property uses the following syntax:

TypeValue:*AddressValue*

The AddressEntry **Address** property corresponds to the MAPI property PR_EMAIL_ADDRESS.

Example

```
' Set up a series of object variables
' Set the Folder and Messages variables; from Session_Inbox
    Set objFolder = objSession.Inbox
    Set objMessages = objFolder.Messages
' Set the Message object variable; from Messages_GetFirst()
    Set objOneMsg = objMessages.GetFirst
' Set the Recipients collection variable; from Message_Recipients()
    Set objRecipColl = objOneMsg.Recipients
' Set the Recipient object variable; from Recipients_FirstItem()
    If 0 = objRecipColl.Count Then
        MsgBox "No recipients in the list"
        Exit Function
    End If
    iRecipCollIndex = 1
    Set objOneRecip = objRecipColl.Item(iRecipCollIndex)
' set the AddressEntry object variable; from Recipient_AddressEntry()
    Set objAddrEntry = objOneRecip.AddressEntry
' from Util_CompareFullAddressParts()
' display the values
    strMsg = "Recipient full address = " & objOneRecip.Address
    strMsg = strMsg & "; AddressEntry type = " & objAddrEntry.Type
    strMsg = strMsg & "; AddressEntry address = " & objAddrEntry.Address
    MsgBox strMsg
```

See Also

Address Property (Recipient Object)

Delete Method (AddressEntry Object)

Deletes the specified address from the address book.

Note OLE Messaging Library version 1.0 supports the **Delete** method only for the Personal Address Book.

Syntax

objAddressEntry.**Delete()**

Parameters

objAddressEntry
Required. The AddressEntry object.

Comments

The **Delete** method fails if both the **Address** and **ID** properties are empty.

Example

```
Function AddressEntry_Delete()
    ' error handling here...
    If objAddrEntry Is Nothing Then
        MsgBox "must select an AddressEntry object"
        Exit Function
    End If
    objAddrEntry.Delete
    Set objAddrEntry = Nothing
    Exit Function
    ' error handling
End Function
```

See Also

Add Method (Recipients Collection)

Details Method (AddressEntry Object)

Displays a dialog box that provides detailed information about an AddressEntry object.

Syntax

objAddressEntry.**Details([***parentWindow***])**

Parameters

objAddressEntry
Required. The AddressEntry object.

parentWindow
Optional. Long. The parent window handle for the details dialog box. A value of **0** (the default, when no value is supplied) specifies a modal dialog box.

Comments

The dialog box always contains at least the display name and address of the address entry. For AddressEntry objects, the method fails if both the **Address** and **ID** properties are empty.

The following methods can also invoke MAPI dialog boxes: **Delete** and **Details** methods (AddressEntry Object), **Options** and **Send** methods (Message object), **Resolve** method (Recipient object and Recipients collection), **AddressBook** and **Logon** methods (Session object).

See Also

Update Method (AddressEntry Object)

ID Property (AddressEntry Object)

Returns the unique ID of the object as a string. Read-only.

Syntax

objAddressEntry.**ID**

Data Type

String

Comments

You can use the AddressEntry object's **ID** property as a parameter to the Recipient object's **Add** method.

MAPI systems assign a permanent, unique **ID** string when an object is created. These IDs do not change from one MAPI session to another.

The **ID** property corresponds to the MAPI property PR_ENTRYID.

Example

This example copies information from an AddressEntry object to a Recipient object:

```
' Function: Recipients_Add_EntryID
' Purpose: Add a new recipient to the collection using AddressEntry ID
Function Recipients_Add_EntryID()
Dim strID As String      ' ID from Message.Sender
Dim strName As String    ' name from Message.Sender
Dim objNewMsg As Object  ' new msg; set its recipient using ID
Dim objNewRecip As Object ' Recipient of new message, set from ID, name
    ' error handling
    strID = objOneMsg.Sender.Id   'Address Entry object ID
    strName = objOneMsg.Sender.Name
    Set objNewMsg = objSession.Outbox.Messages.Add
    If objNewMsg Is Nothing Then
        MsgBox "Could not create a new message"
        Exit Function
    End If
    objNewMsg.Subject = "Sample message from OLE Messaging 1.0"
    objNewMsg.Text = "Called Recipients.Add method w/ entryID parameter"
    Set objNewRecip = objNewMsg.Recipients.Add( _
                      entryID:=strID, _
                      Name:=strName)
    If objNewRecip Is Nothing Then
        MsgBox "Could not create a new recipient"
        Exit Function
    End If
    objNewMsg.Update
    objNewMsg.Send showDialog:=False
    MsgBox "Created a new message in the Outbox and sent it"
    Exit Function
    ' error handling
End Function
```

See Also

Add Method (Recipients Collection)

Name Property (AddressEntry Object)

Returns or sets the display name or alias of the AddressEntry object as a string. Read-write.

Syntax

objAddressEntry.**Name**

Data Type

String

Comments

The AddressEntry object is typically used as a copy of valid addressing information obtained from the Address Book after you have called the Recipient object's **Resolve** method.

When you obtain the AddressEntry object in this context, you should not modify its properties. To request resolution of a display name, use the Recipient object's **Name** property instead. Set the Recipient **Name** property and call the Recipient object's **Resolve** method.

The **Name** property corresponds to the MAPI property PR_DISPLAY_NAME.

Example

```
' for values of variables, see AddressEntry Address property
' Recipient and AddressEntry display names are the same
    strMsg = "Recipient name = " & objOneRecip.Name
    strMsg = strMsg & "; AddressEntry name = " & objAddrEntry.Name
    MsgBox strMsg
```

See Also

Recipient Object

Resolve Method (Recipient Object)

Using Addresses

Type Property (AddressEntry Object)

Specifies the address type, such as "SMTP," "Fax," or "X.400." Read-write.

Syntax

objAddressEntry.**Type**

Data Type

String

Comments

The AddressEntry object's **Type** property specifies the address type. This is usually a tag referring to the messaging system that routes messages to this address, such as "SMTP" or "Fax."

The AddressEntry object's **Address** and **Type** properties combine to form the *full address*, the complete messaging address that appears in the Recipient object's **Address** property. The Recipient **Address** uses the following syntax:

TypeValue:*AddressValue*

The **Type** property corresponds to the MAPI property PR_ADDRTYPE.

Example

See the example for the AddressEntry object **Address** property.

See Also

Address Property (AddressEntry Object)

Address Property (Recipient Object)

Update Method (AddressEntry Object)

Save AddressEntry object changes in the MAPI system.

Syntax

objAddressEntry.**Update**

Parameters

objAddressEntry
Required. The AddressEntry object.

Comments

Changes to objects are not permanently saved in the MAPI system until you use the **Update** method.

Example

The following example changes the display name for the valid AddressEntry address:

```
' Function: AddressEntry_Update
' Purpose: Demonstrate the Update method
'    (Note: OLE Messaging v1.0 only affects the PAB)
Function AddressEntry_Update()
Dim objRecipColl As Object   ' Recipients collection
Dim objNewRecip As Object    ' New recipient

    ' error handling omitted...
    Set objRecipColl = objSession.AddressBook
    If objRecipColl Is Nothing Then
        MsgBox "must select someone from the address book"
        Exit Function
    End If
    Set objNewRecip = objRecipColl.Item(1)
    With objNewRecip.AddressEntry
        .Name = .Name & " the Magnificent"
        .Type = "X.500"   ' you can also change the Type
        .Update
    End With
    MsgBox "Updated an address entry name: " & _
            objNewRecip.AddressEntry.Name
    Exit Function
    ' error handling omitted
End Function
```

See Also

Recipient Object

Attachment Object

The Attachment object represents a document that is an attachment of a message.

Properties

Property name	Type	Access
Application	String	Read-only
Class	Long	Read-only
Index	Long	Read-only
Name	String	Read-write
Parent	Object	Read-only
Position	Long	Read-write
Session	Session object	Read-only
Source	String	Read-write
Type	Long	Read-write

Methods

Method name	Parameters
Delete	(none)
ReadFromFile	fileName as String
WriteToFile	fileName as String

See Also

Attachments Collection

Delete Method (Attachment Object)

Deletes the attachment.

Syntax

objAttachment.**Delete()**

Parameters

objAttachment
Required. The Attachment object.

Comments

The Attachment object is set to **Nothing** and is removed from memory, but the change is not permanent until you use the **Update**, **Send**, or **Delete** method on the parent Message object.

See Also

Delete Method (Attachments Collection)

Index Property (Attachment Object)

Returns the index number for the Attachment object within the Attachments collection. Read-only.

Syntax

objAttachment.**Index**

Data Type

Long

Comments

The Index indicates the attachment's position within the parent Attachments collection.

Example

```
Function Attachments_GetByIndex()
Dim lIndex As Long
Dim objOneAttach As Object  ' assume valid attachment
    ' set error handler here
    If objAttachColl Is Nothing Then
        MsgBox "must select an Attachments collection"
        Exit Function
    End If
    If 0 = objAttachColl.Count Then
        MsgBox "must select collection with 1 or more attachments"
        Exit Function
    End If
    ' prompt user for index; for now, use 1
    Set objOneAttach = objAttachColl.Item(1)
    MsgBox "Selected attachment 1: " & objOneAttach.Name
    lIndex = objOneAttach.Index  ' save index to retrieve this later
    ' ...get same attachment object later
    Set objOneAttach = objAttachColl.Item(lIndex)
    If objOneAttach Is Nothing Then
        MsgBox "Error, could not reselect the attachment"
    Else
        MsgBox "Reselected attachment " & lIndex & _
            " using index: " & objOneAttach.Name
    End If
    Exit Function
```

See Also

Attachments Collection

Item Property (Attachments Collection)

Name Property (Attachment Object)

Returns or sets the display name of the Attachment object as a string. Read-write.

Syntax

objAttachment.**Name**

Data Type

String

Comments

The **Name** property corresponds to the MAPI property PR_ATTACH_FILENAME.

Example

See the example for the Attachment Object **Index** property.

See Also

Attachment Object

Position Property (Attachment Object)

Returns or sets the position of the attachment within the body text of the message. Read-write.

Syntax

objAttachment.**Position**

Data Type

Long

Comments

The **Position** property is a long integer describing where the attachment should be placed in the message body. Applications cannot place two attachments in the same location within a message, and attachments cannot be placed beyond the end of the message body. If **Position** is positive, the attachment replaces the character found at that position in the **Text** property of the Message object. If **Position** is zero, the attachment is added to the message, but is not visible in the message body. **Position** cannot be a negative value.

The **Position** property corresponds to the MAPI property PR_RENDERING_POSITION.

Example

```
' from the function Attachments_Add()
    Set objAttach = objAttachColl.Add         ' add an attachment
    With objAttach
        .Type = mapiFileLink
        .Position = 0   ' place at end of message
        .Source = "\\server\bitmaps\honey.bmp"  ' UNC name
    End With
    ' must update the message to save the new info
    objOneMsg.Update   ' update the message
    MsgBox "Added an attachment of type mapiFileLink"
```

See Also

Add Method (Attachments Collection)

Text Property (Message Object)

ReadFromFile Method (Attachment Object)

Loads the contents of an attachment from a file.

Syntax

objAttachment.**ReadFromFile(***fileName***)**

Parameters

objAttachment
Required. The Attachment object.

fileName
Required. The full path and file name to read. For example, C:\DOCUMENT\BUDGET.XLS.

Comments

The **ReadFromFile** method replaces the existing contents of the Attachment object, if any.

The **ReadFromFile** method operates slightly differently, depending on the value of the Attachment **Type** property. The following table describes its operation:

Attachment Type property	**ReadFromFile operation**
mapiFileData	Copies the contents of the specified file to the attachment.
mapiFileLink	(Not supported)
mapiOLE	The specified file must be a valid OLE docfile, such as a file previously written by the **WriteToFile** method with a **mapiOLE** type setting.

The term "OLE docfile format" indicates that the file is written by an application such as Microsoft Word 6.0 that writes files using the OLE **IStorage** and **IStream** interfaces.

Note that **ReadFromFile** does not support **mapiFileLink** attachments.

See Also

Add Method (Attachments Collection)

Type Property (Attachment Object)

WriteToFile Method (Attachment Object)

Source Property (Attachment Object)

For **mapiFileLink** attachments, returns or sets the full path name of the attachment data file. For **mapiOLE** attachments, returns or sets the OLE class name for the attachment. Read-write.

Syntax

objAttachment.**Source**

Data Type

String

Comments

The value of the **Source** property depends on the value of the **Type** property, as described in the table below:

Type property	**Source property**
mapiFileData	Not used; contains an empty string.
mapiFileLink	Specifies a full path name in a universal naming convention (UNC) format, such as \ \SALES\INFO\PRODUCTS\NEWS.DOC.
mapiOLE	Specifies the registered OLE class name of the attachment, such as "Word.Document" or "PowerPoint.Show."

The UNC format is suitable for sending attachments to recipients who have access to a common file server.

The **Source** property corresponds to the MAPI property PR_ATTACH_PATHNAME.

Example

```
' from the function Attachments_Add()
    Set objAttach = objAttachColl.Add         ' add an attachment
    With objAttach
        .Type = mapiFileLink
        .Position = 0   ' place at end of message
        .Source = "\\server\bitmaps\honey.bmp"  ' UNC name
    End With
    ' must update the message to save the new info
    objOneMsg.Update   ' update the message
    MsgBox "Added an attachment of type mapiFileLink"
```

See Also

Add Method (Attachments Collection)

Type Property (Attachment Object)

Type Property (Attachment Object)

Describes the attachment type. Read-write.

Syntax

objAttachment.**Type**

Data Type

Long

Comments

Three attachment types are supported:

Type property	Value	Description
mapiFileData	1	Attachment is the contents of a file. (Default value.)
mapiFileLink	2	Attachment is a link to a file.
mapiOLE	3	Attachment is an OLE object.

The value of the **Type** property determines the valid values for the **Source** property.

The Attachment object **Type** property corresponds to the MAPI property PR_ATTACH_METHOD.

Example

```
' from the function Attachments_Add()
    Set objAttach = objAttachColl.Add         ' add an attachment
    With objAttach
        .Type = mapiFileLink
        .Position = 0   ' place at end of message
        .Source = "\\server\bitmaps\honey.bmp"  ' UNC name
    End With
    ' must update the message to save the new info
    objOneMsg.Update   ' update the message
    MsgBox "Added an attachment of type mapiFileLink"
```

See Also

Add Method (Attachments Collection)

ReadFromFile Method (Attachment Object)

Source Property (Attachment Object)

WriteToFile Method (Attachment Object)

WriteToFile Method (Attachment Object)

Saves the attachment to a file in the file system.

Syntax

objAttachment.**WriteToFile(***fileName***)**

Parameters

objAttachment
Required. The Attachment object.

fileName
Required. String. The full path and file name for the saved attachment. For example, "C:\DOCUMENT\BUDGET.XLS."

Comments

The **WriteToFile** method operates slightly differently, depending on the value of the Attachment **Type** property. The following table describes its operation:

Attachment Type property	**WriteToFile operation**
mapiFileData	Copies the contents of the specified file to the attachment.
mapiFileLink	(Not supported)
mapiOLE	Writes the file as an OLE docfile format.

Note that **WriteToFile** does not support **mapiFileLink** attachments.

See Also

ReadFromFile Method (Attachment Object)

Attachments Collection Object

The Attachments collection contains one or more Attachment objects.

The Attachments collection is considered a *small collection*, which means that it supports count and index values that let you access individual attachment objects through the **Item** property.

Properties

Property name	Type	Access
Application	String	Read-only
Class	Long	Read-only
Count	Long	Read-only
Item	Object	Read-only
Parent	Object	Read-only
Session	Session object	Read-only

Methods

Method name	Parameters
Add	(optional) name as String, (optional) position as Long, (optional) type as Long, (optional) source as String
Delete	(none)

See Also

Attachment Object

OLE Messaging Object Collections

Add Method (Attachments Collection)

Creates a new Attachment object in the Attachments collection.

Syntax

Set *objAttachment* = *objAttachColl*.**Add(** [*name, position, type, source*] **)**

Parameters

objAttachment
On successful return, contains the new Attachment object.

objAttachColl
Required. The Attachments collection object.

name
Optional. String. The display name of the attachment.

position
Optional. Long. The position of the attachment within the body text of the message.

type
Optional. Long. The type of attachment; either **mapiFileData, mapiFileLink,** or **mapiOLE**.

source
Optional. String. The filename that contains the data for the attachment. The specified filename must be in the appropriate format for the attachment type, specified by the *type* parameter. See the comments below for a complete description.

Comments

The **Add** method parameters correspond to the **Name**, **Position**, **Type**, and **Source** properties of the Attachment object. The *source* parameter is also closely related to the **ReadFromFile** method's *filename* parameter.

You can supply the data for the attachment at the same time that you add the attachment to the collection. The **Add** method operates differently, depending on the value of the *type* parameter. The following table describes its operation:

Value of *type* parameter	**Value of *source* parameter**
mapiFileData	Specifies a full path and file name that contains the data for the attachment. For example, C:\DOCUMENT\BUDGET.XLS.
mapiFileLink	Specifies a full path name in a universal naming convention (UNC) format, such as \\\SALES\INFO\PRODUCTS\NEWS.DOC.
mapiOLE	Specifies a full path and file name to a valid OLE docfile. For example, C:\DOCUMENT\BUDGET2.XLS.

When the *type* parameter has the value **mapiFileLink**, the *source* parameter is a full path name in a universal naming convention (UNC) format. This is suitable for sending attachments to recipients who have access to a common file server.

If you do not specify the *type* and *source* parameters when you call the **Add** method, you must later call the **ReadFromFile** method on the new Attachment object to load the attachment's content.

The **Index** of the new Attachment object equals the new **Count** of the Attachments collection. The attachment is saved in the MAPI system when you **Update** or **Send** the parent Message object.

See Also

Count Property (Attachments Collection)

ReadFromFile Method (Attachment Object)

Type Property (Attachment Object)

Count Property (Attachments Collection)

Returns the number of Attachment objects in the collection. Read-only.

Syntax

objAttachColl.**Count**

Data Type

Long

Example

This example stores in an array the names of all Attachment objects in the collection:

```
' from the sample function, TstDrv_Util_SmallCollectionCount
' objAttachColl is an Attachments collection
x = Util_SmallCollectionCount(objAttachColl)

Function Util_SmallCollectionCount(objColl As Object)
Dim strItemName(100) As String      ' Names of objects in collection
Dim i As Integer                    ' loop counter
    On Error GoTo error_olemsg
    If objColl Is Nothing Then
        MsgBox "Must supply a valid collection object as a parameter"
        Exit Function
    End If
    If 0 = objColl.Count Then
        MsgBox "No items in the collection"
        Exit Function
    End If
    For i = 1 To objColl.Count Step 1
        strItemName(i) = objColl.Item(i).Name
        If 100 = i Then ' max size of string array
            Exit Function
        End If
    Next i
    ' error handling here...
End Function
```

See Also

Item Property (Attachments Collection)

Delete Method (Attachments Collection)

Deletes the entire Attachments collection.

Syntax

object.**Delete()**

Parameters

object
Required. The Attachments collection object.

Comments

The object or collection is set to **Nothing** and it is removed from memory, but the change is not permanent until you use the **Update**, **Send**, or **Delete** method on the parent Message object that contained the deleted Attachments collection.

Be cautious using **Delete** with collections, since the method deletes all member objects within a collection.

See Also

Delete Method (Attachment Object)

Message Object

Item Property (Attachments Collection)

Works like the accessor property to return a single item from a collection. Read-only.

Syntax

objAttachColl.**Item(***index***)**

index
An integer that ranges from 1 to *object*.**Count**, or a string that specifies the name of the object.

Data Type

Object

Comments

The **Item** property works like the accessor property for a collection. For example, the following two lines of code are equivalent:

```
objOneMsg.Attachments.Item(1)
' shorter form:
objOneMsg.Attachments(1)
```

You can usually use the shorter form shown above. However, you must explicitly use the **Item** property when you obtain the collection object from a parameter to a function or subprocedure, as in the following example.

```
Sub UseObject(objAttachColl As Object)
    objAttachColl.Item(1).Name = "Budget.XLS"
End Sub
```

Example

```
' from Util_SmallCollectionCount(objColl As Object)
' This sample obtains the collection as a variable
' so it *must* use the Item property
Dim strItemName(100) as String
    ' error handling omitted from this fragment...
    For i = 1 To objColl.Count Step 1
        strItemName(i) = objColl.Item(i).Name
        If 100 = i Then ' max size of string array
            Exit Function
        End If
    Next i
```

See Also

Count Property (Attachments Collection)

Field Object

A Field represents a custom attribute of a Message object. In effect, the Field object gives you the ability to add properties to a Message.

Properties

Property name	Type	Access
Application	String	Read-only
Class	Long	Read-only
ID	Long	Read-only
Index	Long	Read-only
Name	String	Read-only
Parent	Object	Read-only
Session	Session object	Read-only
Type	Integer	Read-write
Value	Variant	Read-write

Methods

Method name	Parameters
Delete	(none)
ReadFromFile	filename as String
WriteToFile	filename as String

Comments

You can add additional properties tailored for your specific application with the Field object and the Fields collection object. Before adding a field for a message or folder, review the properties that are already provided by the OLE Messaging Library. Many of the common attributes are already offered. For example, **Subject** and **Priority** are already defined as Message object properties.

Note that MAPI properties are unnamed when they are accessed in Field objects. For these MAPI properties, the **Name** property is an empty string.

See Also

Fields Collection

Delete Method (Field Object)

Deletes the Field object.

Syntax

object.**Delete()**

Parameters

object
Required. The Field object.

Comments

The object or collection is set to **Nothing** and it is removed from memory, but the change is not permanent until you use the **Update**, **Send**, or **Delete** method on the parent object (either the parent Folder or Message object) that contained the deleted Field object.

See Also

Add Method (Fields Collection)

ID Property (Field Object)

Returns the unique ID of the object as a long integer. Read-only.

Syntax

object.**ID**

Data Type

Long

Comments

The Field object **ID** property is unique among ID properties supported in the OLE Messaging Library. The Field object ID is a long integer that corresponds to a MAPI property tag value. All other ID properties are strings.

Example

```
' The ID property is a long value, not a string
' fragment from the function Field_ID()
'   verify that objOneField is valid, then access
    MsgBox "ID is high-order word: 0x" & Hex(objOneField.Id)
```

See Also

Type Property (Field Object)

Value Property (Field Object)

Index Property (Field Object)

Returns the index number of this Field object within the Fields collection. Read-only.

Syntax

object.**Index**

Data Type

Long

Example

```
' set up a variable as an index to access a small collection
' fragment from the functions Fields_FirstItem, Fields_NextItem
    If objFieldsColl Is Nothing Then
        MsgBox "must first select a Fields collection"
        Exit Function
    End If
    If 0 = objFieldsColl.Count Then
        MsgBox "No fields in the collection"
        Exit Function
    End If
' Fragment from Fields_FirstItem
    iFieldsCollIndex = 1
    Set objOneField = objFieldsColl.Item(iFieldsCollIndex)
    ' verify that the Field object is valid...
' Fragment from Fields_NextItem
    If iFieldsCollIndex >= objFieldsColl.Count Then
        iFieldsCollIndex = objFieldsColl.Count
        MsgBox "Already at end of Fields collection"
        Exit Function
    End If
    iFieldsCollIndex = iFieldsCollIndex + 1
    Set objOneField = objFieldsColl.Item(iFieldsCollIndex)
    ' verify that the Field object is valid...
```

See Also

Count Property (Fields Collection)

Fields Collection

Name Property (Field Object)

Returns the name of the field as a string. Read-only.

Syntax

object.**Name**

Data Type

String

Comments

The **Name** property is read-only. You set the name of the Field object at the time you create it, when you call the Fields collection **Add** method.

Note that Field objects used to access MAPI properties do not have names. Names can appear only on the custom properties that you create. For more information, see the Item property documentation for the Fields collection.

Example

```
' fragment from Fields_Add
Dim objNewField As Object ' new Field object
    Set objNewField = objFieldsColl.Add( _
                      Name:="Keyword", _
                      Class:=vbString, _
                      Value:="Peru")
    If objNewField Is Nothing Then
        MsgBox "could not create new Field object"
        Exit Function
    End If
    cFields = objFieldsColl.Count
    MsgBox "new Fields collection count = " & cFields
' fragment from Field_Name; modified to use objNewField for active Field
    If "" = objNewField.Name Then
        MsgBox "Field has no name; ID = " & objNewField.Id
    Else
        MsgBox "Field name = " & objNewField.Name
    End If
```

See Also

Add Method (Fields Collection)

ReadFromFile Method (Field Object)

Loads the value of a string or binary field from the specified file.

Syntax

object.**ReadFromFile(***fileName***)**

Parameters

object
Required. The Field object.

fileName
Required. The full path and file name to read. For example, C:\DOCUMENT\BUDGET.XLS.

Comments

The **ReadFromFile** method reads the string or binary value from the specified filename and stores it as the value of the Field object. It replaces any previously existing value for the field.

Note that **ReadFromFile** is not supported for simple types, such as Integer, Long, and Boolean. Visual Basic provides common functions to read and write these base types to and from files. The **ReadFromFile** method fails if the **Type** property of the Field is not a string or binary type.

Note that some binary types are converted to a hex string format when they are stored as Field values. Comparison operations on the **Value** property and the actual contents of the file can return "not equal", even though the values are equivalent.

See Also

WriteToFile Method (Field Object)

Type Property (Field Object)

Returns the data type of the Field object. Read-only.

Syntax

object.**Type**

Data Type

Integer

Comments

The **Type** property specifies the data type of the Field object and determines the range of valid values that may be supplied for the **Value** property. You can set the value of the **Type** property by calling the Fields collection **Add** method.

Valid data types are described in the following table:

Type	Description	Numeric value	OLE variant type	MAPI Property type
vbNull	Null	1	VT_NULL	PT_NULL
vbInteger	Integer	2	VT_I2	PT_I2
vbLong	Long integer	3	VT_I4	PT_LONG
vbSingle	4-byte real (floating point)	4	VT_R4	PT_R4
vbDouble	Double (8-byte real)	5	VT_R8	PT_DOUBLE
vbCurrency	Currency	6	VT_CY	PT_CURRENCY PT_I8
vbDate	Date	7	VT_DATE	PT_APPTIME, PT_SYSTIME
vbString	String	8	VT_BSTR	PT_STRING8, PT_UNICODE, PT_CLSID, PT_BINARY
vbBoolean	Boolean	11	VT_BOOL	PT_BOOLEAN
vbDataObject	Data object	13	VT_UNKNOWN	PT_OBJECT
vbBlob	Blob	65	VT_BLOB	PT_BLOB

Example

```
' Fragment from Fields_Add; uses the type "vbString"
    Set objNewField = objFieldsColl.Add( _
                      Name:="Keyword", _
                      Class:=vbString, _
                      Value:="Peru")
'  verify that objNewField is a valid Field object
' Fragment from Field_Type; display the integer type value
```

```
    MsgBox "Field type = " & objOneField.Type
```

See Also

Value Property (Field Object)

Value Property (Field Object)

Returns or sets the value of the Field object. Read-write.

Syntax

objField.**Value**

Data Type

Variant

Comments

The value of the Field object represents a value of the type specified by the **Type** property. For example, when the Field object has the **Type** property **vbBoolean**, the **Value** property can take the value **True** or **False**. When the Field object has the **Type** property **vbInteger**, the **Value** property can contain a short integer.

Example

```
' fragment from function Field_Type()
' after validating the Field object objOneField
    MsgBox "Field type = " & objOneField.Type
' fragment from function Field_Value()...
    MsgBox "Field value = " & objOneField.Value
```

See Also

ID Property (Field Object)

Type Property (Field Object)

WriteToFile Method (Field Object)

Saves the field value to a file in the file system.

Syntax

objField.**WriteToFile(***fileName***)**

Parameters

objField
Required. The Field object.

fileName
Required. The full path and file name for the saved field; for example, "C:\DOCUMENT\BUDGET.XLS."

Comments

The **WriteToFile** method writes the string or binary value of the Field object to the specified filename. It overwrites any existing information in that file.

Note that **WriteToFile** is not supported for simple types, such as Integer, Long, and Boolean. Visual Basic provides common functions to read and write these base types to and from files. The **WriteToFile** method fails if the **Type** property of the Field is not a string or binary type.

Note that some binary types are represented in hex string format by the OLE Messaging Library and written in binary format. Comparison operations on the **Value** property and the actual contents of the file can return "not equal," even though the values are equivalent.

See Also

ReadFromFile Method (Field Object)

Fields Collection Object

The Fields collection represents one or more Field objects. Field objects give you the ability to add custom fields to a message or folder.

The Fields collection is considered a *small collection*, which means that it supports count and index values that let you access individual Field objects through the **Item** property.

Properties

Property name	Type	Access
Application	String	Read-only
Class	Long	Read-only
Count	Long	Read-only
Item	Object	Read-only
Parent	Object	Read-only
Session	Session object	Read-only

Methods

Method name	Parameters
Add	name as String, Class as Long, value as Variant
Delete	(none)

Example

To uniquely identify Field objects in the Fields collection, use the Field object's **Name** property or an index:

```
Set objOneField = objFolder.Fields.Item("BalanceDue")
Set objAnotherField = objMessage.Fields.Item("Keyword")
Set objThirdField = objMessage.Fields.Item(3)
```

See Also

OLE Messaging Object Collections

Add Method (Fields Collection)

Creates a new Field object in the Fields collection.

Syntax

Set *objField* **=** *objFieldsColl*.**Add (***name, Class, value***)**

Note The *Class* parameter may be renamed *type* in future versions of the OLE Messaging library to avoid confusion with the **Class** property.

Parameters

objField
On successful return, contains the new Field object.

objFieldsColl
Required. The Fields collection object.

name
Required. A string that represents the display name of the field.

Class
Required. A constant long integer that represents the data type for the field, such as string or integer. The *Class* parameter represents the same values as the Field object **Type** property. The following types are allowed:

Type property	Description	Numeric value	OLE variant type
vbNull	Null	1	VT_NULL
vbInteger	Integer	2	VT_I2
vbLong	Long integer	3	VT_I4
vbSingle	4-byte real (floating point)	4	VT_R4
vbDouble	Double (8-byte real)	5	VT_R8
vbCurrency	Currency	6	VT_CY
vbDate	Date	7	VT_DATE
vbString	String	8	VT_BSTR
vbBoolean	Boolean	11	VT_BOOL
vbDataObject	Data object	13	VT_UNKNOWN
vbBlob	Blob	65	VT_BLOB

value
Required. Variant. The value of the field, of the data type specified in the *type* parameter.

Comments

The method parameters correspond to the **Name**, **Type**, and **Value** properties of the Field object.

The **Index** of the new Field object equals the new **Count** of the Fields collection. The field is saved in the MAPI system when you **Update** or **Send** the parent object.

Note that when you use the type VbString to represent a PT_BINARY type, you actually supply the value in the form of a string that contains the hex representation of the bytes in the binary object (such as a hex dump of the object).

Example

```
' Fragment from Fields_Add; uses the type "vbString"
    Set objNewField = objFieldsColl.Add( _
                      Name:="Keyword", _
                      Class:=vbString, _
                      Value:="Peru")
'  verify that objNewField is a valid Field object
' Fragment from Field_Type; display the integer type value
    MsgBox "Field type = " & objOneField.Type
```

See Also

Count Property (Fields Collection)

Field Object

Count Property (Fields Collection)

Returns the number of Field objects in the collection. Read-only.

Syntax

objFieldsColl.**Count**

Data Type

Long

Example

This example maintains a global variable as an index into the small collection, and uses the Count property to check its validity.

```
' from Fields_NextItem
' iFieldsCollIndex is an integer used as an index
'   check for empty collection...
'   check index upper bound
    If iFieldsCollIndex >= objFieldsColl.Count Then
        iFieldsCollIndex = objFieldsColl.Count
        MsgBox "Already at end of Fields collection"
        Exit Function
    End If
    ' index is < count; can be incremented by 1
    iFieldsCollIndex = iFieldsCollIndex + 1
    Set objOneField = objFieldsColl.Item(iFieldsCollIndex)
    If objOneField Is Nothing Then
        MsgBox "Error, cannot get this Field object"
        Exit Function
    Else
        MsgBox "Selected field " & iFieldsCollIndex
    End If
```

See Also

Field Object

Delete Method (Fields Collection)

Deletes the Fields collection object.

Syntax

objFieldsColl.**Delete**

Parameters

objFieldsColl
Required. The Fields collection object.

Comments

The object or collection is set to **Nothing** and it is removed from memory, but the change is not permanent until you use the **Update**, **Send**, or **Delete** method on the parent Message object that contained the deleted Fields collection.

Be cautious using **Delete** with collections, since the method deletes all member objects within a collection.

See Also

Field Object

Item Property (Fields Collection)

Works like the accessor property to return a single item from a collection. Read-only.

Syntax

objFieldsColl.**Item(***index***)**

objFieldsColl
Required. Specifies the Fields collection object.

index
Either a long integer or a string. Long integer values less than or equal to 65,535 (&Hffff) are interpreted as indexes into the Fields collection. Long integer values greater than 65,535 are interpreted as MAPI property tag values. The string index is interpreted as the name of the property.

Data Type

Object

Comments

The **Item** property in the Fields collection object allows access to the underlying MAPI properties.

The long value greater than 65,535 represents a *property tag*. A property tag is a 32-bit unsigned integer that contains the property identifier in its high-order 16 bits and the property type (its underlying data type) in the low-order 16 bits. Several macros for C/C++ programmers are available in the MAPI 1.0 SDK to help manipulate the property tag data structure. The macros PROP_TYPE and PROP_ID extract the property type and property identifer from the property tag. The macro PROP_TAG builds the property tag from the provided type and identifier components.

For example, you can use the following function to access the Field object by name.

```
' from the function Fields_ItemByName()
    ' error handling here...
    If objFieldsColl Is Nothing Then
        MsgBox "must first select Fields collection"
        Exit Function
    End If
    Set objOneField = objFieldsColl.Item("Keyword")
    If objOneField Is Nothing Then
        MsgBox "could not select Field object"
        Exit Function
    End If
    If "" = objOneField.Name Then
        MsgBox "Field has no name; ID = " & objOneField.Id
    Else
        MsgBox "Field name = " & objOneField.Name
    End If
```

You can also use the Item property to access MAPI properties. Note that all MAPI properties appear to have no name. For example, you can use the following code to obtain the MAPI property PR_MESSAGE_CLASS:

```
' from the function Fields_Selector()
    ' ... error handling here
    ' you can provide a dialog to allow entry for MAPI proptags
    ' or select property names from a list; for now, hard-coded
```

```
lValue = &h1a001e ' &H1a = PR_MESSAGE_CLASS;
                  ' &H001e = 30 = PT_STRING8
' high-order 16 bits is property id; low-order is property type
Set objOneField = objFieldsColl.Item(lValue)
If objOneField Is Nothing Then
    MsgBox "Could not get the Field using the value " & lValue
    Exit Function
Else
    strMsg = "Used " & lValue & " to access the MAPI property "
    strMsg = strMsg & "PR_MESSAGE_CLASS: type = " & objOneField.Type
    strMsg = strMsg & "; value = " & objOneField.Value
    MsgBox strMsg
End If
```

The built-in MAPI properties are not named properties, so they can only be accessed using the numeric value. They can not be accessed using a string that represents the name. For a complete discussion of MAPI properties and list of MAPI property tag values, see the Microsoft *MAPI Programmer's Reference*.

OLE Messaging version 1.0 does not support multi-valued MAPI properties.

See Also

Customizing A Folder Or Message

Viewing MAPI Properties

Field Object

Folder Object

The Folder object represents a folder or container within the MAPI system. Folders can contain subfolders and messages.

Properties

Property name	Type	Access
Application	String	Read-only
Class	Long	Read-only
Fields	Fields collection object	Read-only
FolderID	String	Read-only
Folders	Folders collection object	Read-only
ID	String	Read-only
MAPIOBJECT	Object	Read-write (Note: Not available to Visual Basic applications)
Messages	Messages collection object	Read-only
Name	String	Read-write
Parent	Object	Read-only
Session	Session object	Read-only
StoreID	String	Read-only

Methods

(None.)

Comments

Changes to the folder are immediately saved to the MAPI system. The Folder object does not have an **Update** method.

The **ID** property is unique and read-only. MAPI assigns a unique **ID** when the Folder object is created. Its value does not change.

Note that the OLE Messaging Library version 1.0 does not support methods to allow you to create new folders. You can access existing folders created by other applications, such as Microsoft Exchange.

See Also

Folders Collection

Fields Property (Folder Object)

Returns a single field (a Field object) or a collection of fields (a Fields collection object) of the Folder object. Read-only.

Syntax

objFolder.**Fields**

objFolder.**Fields(***index***)**

index
Specifies the name of the field or the number of the field.

Data Type

Object

Comments

Fields provide a generic access mechanism that allows Visual Basic programmers to retrieve the value of any property associated with the Folder object using either a name or a property tag. To access using the property tag, use Folder.Fields.Item(*proptag*), where *proptag* is the 32-bit MAPI property tag associated with the property in question, such as PR_MESSAGE_CLASS. To access a Field object using a name, use Folder.Fields.Item(*name*), where name is a string that represents the custom property name.

Example

This example stores in an array the names of all fields of a folder:

```
' many properties are MAPI properties and have no names
' for those properties, display the ID
' fragment from Field_Name
' assume objOneField is a valid Field object
    If "" = objOneField.Name Then
        MsgBox "Field has no name; ID = " & objOneField.Id
    Else
        MsgBox "Field name = " & objOneField.Name
    End If
```

See Also

Field Object

FolderID Property (Folder Object)

Returns the unique ID of this subfolder's parent folder as a string. Read-only.

Syntax

objFolder.**FolderID**

Data Type

String

Comments

MAPI systems assign a permanent, unique ID string when an object is created. These IDs do not change from one MAPI session to another.

With OLE Messaging Library version 1.0, when you follow the parent **FolderID** properties upward, you eventually encounter a folder with its **Name** property set to "Top of Information Store." This represents the top of the tree for the message class, "IPM.Note".

The **FolderID** property corresponds to the MAPI property PR_PARENT_ENTRYID.

Example

```
'          fragment from Session_Inbox
    Set objFolder = objSession.Inbox
'          fragment from Folder_FolderID
    strFolderID = objFolder.FolderID
    MsgBox "Parent Folder ID = " & strFolderID
'          can later restore using objSession.GetFolder(strFolderID)
'          fragment from Session_GetFolder
    If "" = strFolderID Then
        MsgBox ("Must first set folder ID variable; see Folder->ID")
        Exit Function
    End If
    Set objFolder = objSession.GetFolder(strFolderID)
    ' error checking here...
```

See Also

ID Property (Folder Object)

Folders Property (Folder Object)

Specifies a collection of subfolders within the parent folder. Returns a single folder (a Folder object) or a collection of subfolders (a Folders collection). Read-only.

Syntax

objFolder.**Folders**

objFolder.**Folders(***index***)**

index
Specifies the **FolderID** or number of the folder.

Data Type

Object

Example

This example lists all the names of all subfolders of the specified folder:

```
' fragment from Session_Inbox
    Set objFolder = objSession.Inbox
' from TstDrv_Util_ListFolders
   If 2 = objFolder.Class Then  ' verify Folder object
        x = Util_ListFolders(objFolder)  ' use current global folder
   End If

' complete function for Util_ListFolders
Function Util_ListFolders(objParentFolder As Object)
Dim objFoldersColl As Object ' the child Folders collection
Dim objOneSubfolder As Object 'a single Folder object
    ' set up error handler here
    If Not objParentFolder Is Nothing Then
        MsgBox ("Folder name = " & objParentFolder.Name)
        Set objFoldersColl = objParentFolder.Folders
        If Not objFoldersColl Is Nothing Then ' loop through all
            Set objOneSubfolder = objFoldersColl.GetFirst
            While Not objOneSubfolder Is Nothing
                x = Util_ListFolders(objOneSubfolder)
                Set objOneSubfolder = objFoldersColl.GetNext
            Wend
        End If
    End If
    Exit Function
    ' error handler here
End Function
```

See Also

Folders Collection

ID Property (Folder Object)

Returns the unique ID of this Folder object as a string. Read-only.

Syntax

objFolder.**ID**

Data Type

String

Comments

MAPI systems assign a permanent, unique ID string when an object is created. These IDs do not change from one MAPI session to another.

The **ID** property corresponds to the MAPI property PR_ENTRYID.

Example

```
' similar example to FolderID;
' save the current ID and restore using Session.GetFolder
'          fragment from Session_Inbox
    Set objFolder = objSession.Inbox
'          fragment from Folder_FolderID
    strFolderID = objFolder.ID
    MsgBox "Parent Folder ID = " & strFolderID
'          can later restore using objSession.GetFolder(strFolderID)
'          fragment from Session_GetFolder
    If "" = strFolderID Then
        MsgBox ("Must first set folder ID variable; see Folder->ID")
        Exit Function
    End If
    Set objFolder = objSession.GetFolder(strFolderID)
    ' error checking here...
```

See Also

Folders Property (Folder Object)

GetFolder Method (Session Object)

MAPIOBJECT Property (Folder Object)

Returns an **IUnknown** pointer to this Folder object. Not available to Visual Basic applications.

Syntax

objFolder.**MAPIOBJECT**

Data Type

Variant (VT_UNKNOWN)

Comments

The **MAPIOBJECT** property is not available to Visual Basic programs. It is available only to C/C++ programs that use the OLE Messaging Library. The **MAPIOBJECT** property is an **IUnknown** object, which is not supported by Visual Basic. Visual Basic supports **IDispatch** objects. For more information, see the Microsoft *OLE 2 Programmer's Reference*.

See Also

A ShortTour of OLE Automation

How Programmable Objects Work

Messages Property (Folder Object)

Returns a single Message object or a Messages collection object within the folder. Read-only.

Syntax

objFolder.**Messages**

objFolder.**Messages(***ID***)**

ID
Specifies the **ID** of the message.

Data Type

Object

Example

```
' from the QuickStart sample
' use the Messages property of the Outbox folder
    Set objSession = CreateObject("MAPI.Session")
    objSession.Logon
    Set objMessage = objSession.Outbox.Messages.Add
```

See Also

ID Property (Message Object)

Message Object

Messages Collection

Name Property (Folder Object)

Returns or sets the name of the Folder object as a string. Read-write.

Syntax

objFolder.**Name**

Data Type

String

Comments

The **Name** property corresponds to the MAPI property PR_DISPLAY_NAME.

Example

```
Dim objFolder As Object  ' assume valid folder
MsgBox "Folder name = " & objFolder.Name
```

See Also

GetFolder Method (Session Object)

StoreID Property (Folder Object)

Returns the name of the Store object as a string. Read-only.

Syntax

objFolder.**StoreID**

Data Type

String

Comments

The **StoreID** property corresponds to the MAPI property PR_STORE_ENTRYID.

Example

```
' from the sample function Folder_ID
    strFolderID = objFolder.ID
' from the sample function Folder_StoreID
    strFolderStoreID = objFolder.storeID
'  can use these IDs with Session.GetFolder()
'  from the sample function Session_GetFolder
    Set objFolder = objSession.GetFolder(folderID:=strFolderID, _
                                         storeID:=strFolderStoreID)
```

See Also

GetFolder Method (Session Object)

Folders Collection Object

The Folders collection contains one or more Folder objects.

The Folders collection is considered a *large collection*, which means that you must use a Folder ID value or the **Get*** methods to access individual Folder objects within the collection.

Properties

Property name	Type	Access
Application	String	Read-only
Class	Long	Read-only
Parent	Object	Read-only
Session	Session object	Read-only

Methods

Method name	Parameters
GetFirst	(none)
GetLast	(none)
GetNext	(none)
GetPrevious	(none)

Comments

Large collections, such as the Folders collection, do not maintain a count of the number of objects in the collection. Instead you must use the **GetFirst, GetNext, GetLast**, and **GetPrevious** methods to access individual items in the collection. You can also access a specific folder by using the Session object's **GetFolder** method.

Example

To refer to a unique Folder object within the Folders collection, use the collection's **Get*** methods or use the **FolderID** value as the index:

```
' assume objSession valid, user logged on
' fragment from the function Session_GetFolder
    Set objFolderColl = objSession.GetFolder(strFolderID)
' fragment from Folders_Class
    MsgBox "Folders class =" & objFolderColl.Class
' Get a specific folder using an ID
    Set objOneFolder = objFolderColl(strFolderID)
' fragment from Folders_GetFirst
    Set objOneFolder = objFolderColl.GetFirst
' fragment form Folders_GetNext
    Set objOneFolder = objFolderColl.GetNext
```

The following code sample demonstrates the **Get*** methods. The sample assumes that you have three subfolders within your Inbox and three subfolders within your Outbox. After this code runs, the three folders in the Inbox are named Blue, Red, and Orange (in that order), and the three folders in the Outbox are named Gold, Purple, and Yellow (in that order).

```
Dim objSession As Object
Dim objMessage As Object
Dim objFolder As Object
```

```
Set objSession = CreateObject("MAPI.Session")
objSession.Logon "User", "", True
With objSession.Inbox.Folders
   Set objFolder = .GetFirst
   objFolder.Name = "Blue"
   Set objFolder = .GetNext
   objFolder.Name = "Red"
   Set objFolder = .GetLast
   objFolder.Name = "Orange"
End With
With objSession.Outbox.Folders
   Set objFolder = .GetFirst
   objFolder.Name = "Gold"
   Set objFolder = .GetNext
   objFolder.Name = "Purple"
   Set objFolder = .GetLast
   objFolder.Name = "Yellow"
End With
objSession.Logoff
```

See Also

OLE Messaging Object Collections

GetFirst Method (Folders Collection)

Returns the first object in the Folders collection. Returns **Nothing** if no first object exists.

Syntax

Set *objFolder* = *objFoldersColl*.**GetFirst()**

Parameters

objFolder
On successful return, represents the first folder object in the collection.

objFoldersColl
Required. The Folders collection object.

Comments

The **Get*** methods are similar to the **Find*** and **Move*** methods that are used with Microsoft Access databases. The **Get*** methods take a different name from these methods because they use a different syntax.

See Also

Type property (Message object)

Folder Object

GetLast Method (Folders Collection)

Returns the last folder object in the Folders collection. Returns **Nothing** if no last object exists.

Syntax

Set *objFolder* = *objFoldersColl*.**GetLast()**

Parameters

objFolder
On successful return, represents the last folder object in the collection.

objFoldersColl
Required. The Folders collection object.

Comments

The **Get*** methods are similar to the **Find*** and **Move*** methods that are used with Microsoft Access databases. The **Get*** methods take a different name from these methods because they use a different syntax.

See Also

Folder Object

GetNext Method (Folders Collection)

Returns the next object in the Folders collection. Returns **Nothing** if no next object exists, or when already positioned at the end of the folders collection.

Syntax

Set *objFolder* = *objFoldersColl*.**GetNext()**

Parameters

objFolder
On successful return, represents the next folder object in the collection.

objFoldersColl
Required. The Folders collection object.

Comments

The **Get*** methods are similar to the **Find*** and **Move*** methods that are used with Microsoft Access databases. The **Get*** methods take a different name from these methods because they use a different syntax.

See Also

Folder Object

GetPrevious Method (Folders Collection)

Returns the previous object in the Folders collection. Returns **Nothing** if no previous object exists, or when already positioned at the first folder in the collection.

Syntax

Set *objFolder* = *objFoldersColl*.**GetPrevious()**

Parameters

objFolder
On successful return, represents the previous folder object in the collection.

objFoldersColl
Required. The Folders collection object.

Comments

The **Get*** methods are similar to the **Find*** and **Move*** methods that are used with Microsoft Access databases. The **Get*** methods take a different name from these methods because they use a different syntax.

See Also

Folder Object

Message Object

The Message object represents a single message, item, document, or form in a folder.

Properties

Property name	Type	Access
Application	String	Read-only
Attachments	Attachments collection object	Read-only
Class	Long	Read-only
ConversationIndex	String	Read-write
ConversationTopic	String	Read-write
DeliveryReceipt	Boolean	Read-write
Encrypted	Boolean	Read-write
Fields	Fields collection object	Read-only
FolderID	String	Read-only
ID	String	Read-only
Importance	Integer	Read-write
MAPIOBJECT	(Not for use with VB)	Read-write (Note: Not available to Visual Basic applications)
Parent	Object	Read-only
ReadReceipt	Boolean	Read-write
Recipients	Recipients object	Read-only
Sender	AddressEntry object	Read-only
Sent	Boolean	Read-write
Session	Session object	Read-only
Signed	Boolean	Read-write
Size	Long	Read-only
StoreID	String	Read-only
Subject	String	Read-write
Submitted	Boolean	Read-write
Text	String	Read-write
TimeReceived	Variant (Date/Time)	Read-write
TimeSent	Variant (Date/Time)	Read-write
Type	Integer	Read-write
Unread	Boolean	Read-write

Methods

Method name	Parameters
Delete	(none)
Options	(optional) parentWindow as Long
Send	(optional) saveCopy as Boolean, (optional) showDialog as Boolean, (optional) parentWindow

	as Long
Update	(none)

Comments

The **ID** property is unique and read-only. MAPI assigns a unique **ID** to each Message object and its value does not change.

You can save the **ID** property and use it to retrieve the message later using the Session object's **GetMessage** method.

Example

```
' save the message ID, use it to later restore the message
' from the sample function Message_ID
    strMessageID = objMessage.ID
' from the sample function Session_GetMessage
    Set objMessage = objSession.GetMessage(strMessageID)
```

See Also

GetMessage Method (Session Object)

Messages Collection

Messages Property (Folder Object)

Attachments Property (Message Object)

Returns a single Attachment object or an Attachments collection. Read-only.

Syntax

object.**Attachments**

object.**Attachments(***index***)**

index
Specifies the number of the attachment within the Attachments collection.

Example

This example stores in an array the names of all attachments of a message:

```
' from the sample function Message_Attachments
    Set objAttachColl = objOneMsg.Attachments
    If objAttachColl Is Nothing Then
        MsgBox "unable to set Attachments collection"
        Exit Function
    Else
        MsgBox "Attachments count for this msg: " & objAttachColl.Count
        iAttachCollIndex = 0    ' reset global index variable
    End If
' from the sample function Attachments_FirstItem
    iAttachCollIndex = 1
    Set objAttach = objAttachColl.Item(iAttachCollIndex)
```

See Also

Attachment Object

Attachments Collection

ConversationIndex Property (Message Object)

Specifies the index to the conversation thread of the message. Read-write.

Syntax

object.**ConversationIndex**

Data Type

String

Comments

The **ConversationIndex** property is a string that represents a hexadecimal number. Valid characters within the string include the numbers 0 through 9 and the letters A through F (uppercase or lowercase).

A conversation is a group of related messages that have the same **ConversationTopic** property value. In a discussion application, for example, users can save original messages and response messages. Messages can be tagged with the **ConversationIndex** property so that users can group messages by conversation.

You can use your own convention to decide how this index should be used. However, it is recommended that you adopt the same convention that is used by the Microsoft Exchange message viewer, so that you can use that viewer's user interface to show the relationships between messages in a conversation.

By convention, Microsoft Exchange uses **ConversationIndex** values that represent concatenated time stamp values. The first time stamp in the string represents the original message. When a new message represents a reply to a conversation message, it copies the **ConversationIndex** string of the message it is replying to, and then appends a time stamp value to the end of the string. The new string value is used as the **ConversationIndex** value of the new message.

When you use this convention, you can see relationships among messages when you sort the messages by **ConversationIndex** values.

The **ConversationIndex** property corresponds to the MAPI property PR_CONVERSATION_INDEX.

Example

The following example takes advantage of an OLE API function that is available on computers that run the OLE Messaging Library. The **CoCreateGUID** function returns a value that consists of a time stamp and a machine identifier; this sample code saves the part that contains the time stamp.

```
' declarations section
Type GUID  ' global unique identifier; contains a time stamp
    Guid1 As Long
    Guid2 As Long
    Guid3 As Long
    Guid4 As Long
End Type
' function appears in OLE32.DLL on Windows/NT and Windows 95
Declare Function CoCreateGuid Lib "COMPOBJ.DLL" (pGuid As GUID) As Long
Global Const S_OK = 0   ' return value from CoCreateGuid

Function Util_GetEightByteTimeStamp() As String
Dim lResult As Long
Dim lGuid As GUID
```

```
    ' Exchange conversation is a unique 8-byte value
    ' Exchange client viewer sorts by concatenated properties
    On Error GoTo error_olemsg

    lResult = CoCreateGuid(lGuid)
    If lResult = S_OK Then
        Util_GetEightByteTimeStamp = _
            Hex$(lGuid.Guid1) & Hex$(lGuid.Guid2)
    Else
        Util_GetEightByteTimeStamp = "00000000"   ' zeroes
    End If
    Exit Function

error_olemsg:
    MsgBox "Error " & Str(Err) & ": " & Error$(Err)
    Util_GetEightByteTimeStamp = "00000000"
    Exit Function
End Function

Function Util_NewConversation()
Dim i As Integer
Dim objNewMsg As Object      ' new message object
Dim strNewIndex As String    ' value for ConversationIndex
' ... error handling...
    Set objNewMsg = objSession.Outbox.Messages.Add
' ... error handling...
    With objNewMsg
        .Subject = "used space vehicle wanted"
        .ConversationTopic = .Subject
        .ConversationIndex = Util_GetEightByteTimeStamp() ' utility
        .Text = "Wanted: Apollo or Mercury spacecraft with low mileage."
           ' or you could pick the public folder from the address book
        Set objOneRecip = .Recipients.Add(Name:="Car Ads", Type:=mapiTo)
        If objOneRecip Is Nothing Then
            MsgBox "Unable to create the public folder recipient"
            Exit Function
        End If
        .Recipients.Resolve
        .Update
        .Send showDialog:=False
    End With
End Function
```

A subsequent reply to a message should copy the **ConversationTopic** property and append its own time stamp to the original message's time stamp, as shown in the following example:

```
Function Util_ReplyToConversation()
Dim objPublicFolder As Object
Dim i As Integer
Dim objOriginalMsg As Object ' original message in public folder
Dim objNewMsg As Object      ' new message object for reply
Dim strPublicFolderID As String ' ID for public folder

    Set objNewMsg = objSession.Outbox.Messages.Add
'   error checking...obtain objOriginalMsg and check that it is valid
```

```
    With objNewMsg
        .Text = "How about a slightly used Gemini?"   ' new text
        .Subject = objOriginalMsg.Subject   ' copy original properties
        .ConversationTopic = objOriginalMsg.ConversationTopic
        ' append time stamp; compatible with Microsoft Exchange client
        .ConversationIndex = objOriginalMsg.ConversationIndex & _
                             Util_GetEightByteTimeStamp() ' append new
        ' message was sent to a public folder so can copy recipient
        Set objOneRecip = .Recipients.Add( _
                          Name:=objOriginalMsg.Recipients.Item(1).Name, _
                          Type:=mapiTo)
       ' ...more error handling
        .Recipients.Resolve
        .Update
        .Send showDialog:=False
    End With
' ... error handling
End Function
```

See Also

ConversationTopic Property (Message Object)

Working With Conversations

ConversationTopic Property (Message Object)

Specifies the name of the conversation thread. Read-write.

Syntax

object.**ConversationTopic**

Data Type

String

Comments

A conversation is a group of related messages. The **ConversationTopic** property is the string that describes the overall topic of the conversation. To be defined as messages within the same conversation, the messages must have the same value in their **ConversationTopic** property. The **ConversationIndex** property represents an index that indicates a sequence of messages within that conversation.

When you start an initial message, set the **ConversationTopic** property to an appropriate value that will apply to all messages within the conversation. For many applications, the message **Subject** property is appropriate.

Note that the OLE Messaging Library does not automatically copy the **ConversationTopic** property to other messages. When your application manages messages that represent replies to an original message, you should set the **ConversationTopic** property to the same value as the original message.

To change the **ConversationTopic** for all messages in a conversation thread, you must change the property within each message in that thread.

The **ConversationTopic** property corresponds to the MAPI property PR_CONVERSATION_TOPIC.

Example

See the example for the **ConversationIndex** property.

See Also

ConversationIndex Property (Message Object)

Working With Conversations

Delete Method (Message Object)

Deletes the Message object.

Syntax

object.**Delete**

Parameters

object
Required. The Message object.

Comments

Moves the deleted message to the Deleted Messages folder, if the user has enabled this option. If the message is deleted from the Deleted Messages folder, the **Delete** method permanently deletes it from the system and it cannot be recovered.

See Also

Delete Method (Messages Collection)

DeliveryReceipt Property (Message Object)

True if a delivery-receipt notification message is requested. Read-write.

Syntax

object.**DeliveryReceipt**

Data Type

Boolean

Comments

Set the **DeliveryReceipt** property to **True** to obtain a message when the recipients receive the message. The default setting for OLE Messaging is False. The **DeliveryReceipt** property is completely independent of the settings from Microsoft Exchange.

The **DeliveryReceipt** property corresponds to the MAPI property PR_ORIGINATOR_DELIVERY_REPORT_REQUESTED.

See Also

Making Sure The Message Gets There

ReadReceipt Property (Message Object)

Encrypted Property (Message Object)

True if the message has been encrypted. Read-write.

Syntax

object.**Encrypted**

Data Type

Boolean

Comments

The OLE Messaging Library does not encrypt or digitally sign the message. The **Encrypted** property is dependent upon the messaging or information store provider.

The **Encrypted** property corresponds to the MAPI property PR_SECURITY.

See Also

Securing Messages

Signed Property (Message Object)

Fields Property (Message Object)

Returns a single field (a Field object) or a collection of fields (a Fields collection object) of the Message object. Read-only.

Syntax

objMessage.**Fields**

objMessage.**Fields(***index***)**

index
Specifies the name of the field or the number of the field.

Data Type

Field object or Fields collection

Example

```
' from Message_Fields
    Set objFieldsColl = objOneMsg.Fields
' from Fields_FirstItem
    iFieldsCollIndex = 1
    Set objOneField = objFieldsColl.Item(iFieldsCollIndex)
    If objOneField Is Nothing Then
        MsgBox "error, cannot get this Field object"
    Else
        MsgBox "Selected field " & iFieldsCollIndex
    End If
```

See Also

Fields Collection

FolderID Property (Message Object)

Returns the unique ID of the Folder in which the Message resides. Read-only.

Syntax

objMessage.**FolderID**

Data Type

String

Comments

Save the folder ID to retrieve the folder at a later time using the Session object's **GetFolder** method.

MAPI systems assign a permanent, unique ID string when an object is created. These IDs do not change from one MAPI session to another.

The **FolderID** property corresponds to the MAPI property PR_PARENT_ENTRYID.

See Also

GetFolder Method (Session Object)

ID Property (Message Object)

Returns the unique ID of the object as a string. Read-only.

Syntax

objMessage.**ID**

Data Type

String

Comments

MAPI systems assign a permanent, unique ID string when an object is created. These IDs do not change from one MAPI session to another.

The **ID** property corresponds to the MAPI property PR_ENTRYID.

Example

```
'     Save id of last message accessed; use at startup
'     from the sample function Message_ID
    strMessageID = objOneMsg.Id

' ... on shutdown, save the ID to storage
' ... on startup, get the ID from storage and restore
'     from the sample function Session_GetMessage
    Set objOneMsg = objSession.GetMessage(strMessageID)
```

See Also

GetMessage Method (Session Object)

Importance Property (Message Object)

Returns or sets the importance of the message as one of **mapiNormal** (the default), **mapiLow,** or **mapiHigh**. Read-write.

Syntax

object.**Importance**

Data Type

Long (Enumeration)

Comments

The following values are defined:

Constant	Value	Description
mapiLow	0	Low priority
mapiNormal	1	Normal priority (default)
mapiHigh	2	High priority

The **Importance** property corresponds to the MAPI property PR_IMPORTANCE.

Example

This example sets the importance of a message as "high:"

```
' from the sample function QuickStart:
    Set objMessage = objSession.Outbox.Messages.Add
    '  error checking here to verify the message was created...
    objMessage.Subject = "Gift of droids"
    objMessage.Text = "Help us, Obi-wan. You are our only hope."
    objMessage.Importance = mapiHigh
    objMessage.Send
```

See Also

Send Method (Message Object)

MAPIOBJECT Property (Message Object)

Returns an **IUnknown** pointer to this Message object. Not available to Visual Basic applications.

Syntax

object.**MAPIOBJECT**

Data Type

Variant (VT_UNKNOWN)

Comments

The **MAPIOBJECT** property is not available to Visual Basic programs. It is available only to C/C++ programs that use the OLE Messaging Library. The **MAPIOBJECT** property is an **IUnknown** object, which is not supported by Visual Basic. Visual Basic supports **IDispatch** objects. For more information, see the Microsoft *OLE 2 Programmer's Reference*.

See Also

A Short Tour of OLE Automation

How Programmable Objects Work

Options Method (Message Object)

Displays a message options dialog box where the user can change the submission options for a message.

Syntax

object.**Options([***parentWindow***])**

Parameters

object
Required. The Message object.

parentWindow
Optional. Long. The parent window handle for the options dialog box. A value of **0** (the default) specifies an application-modal dialog box.

Comments

The options are provider-specific and are registered by the provider. Providers are not required to register option sheets. When providers do not register options, the Message object **Options** method does nothing and appears to succeed (it does not raise an exception).

Per-message options are properties of a message that control its behavior after submission. The per-message options are part of the message envelope, not its content.

The following methods can also invoke MAPI dialog boxes: **Delete** and **Details** methods (AddressEntry object), **Send** method (Message object), **Resolve** method (Recipient object and Recipients collection), **AddressBook** and **Logon** methods (Session object).

See Also

Send Method (Message Object)

ReadReceipt Property (Message Object)

True if a read-receipt notification message is requested. Read-write.

Syntax

object.**ReadReceipt**

Data Type

Boolean

Comments

The **ReadReceipt** property corresponds to the MAPI property PR_READ_RECEIPT_REQUESTED.

See Also

DeliveryReceipt Property (Message Object)

Making Sure The Message Gets There

Recipients Property (Message Object)

Returns a single Recipient object or a Recipients collection object. Read-only.

Syntax

Set *objRecipColl* **=** *objMessage.***Recipients**

Set *objOneRecip* = *objMessage.***Recipients(***index***)**

objRecipColl
Object. A Recipients collection object.

objMessage
Object. The Message object.

objOneRecip
Object. A single Recipient object.

index
Long. Specifies the number of the recipient within the Recipients collection. Ranges from 1 to the value specified by the Recipients collection **Count** property.

Data Type

Object

Example

This example copies each of the recipients from the original message objOneMsg to the copy of the message, objCopyMsg:

```
' from the sample function Util_CopyMessage
    For i = 1 To objOneMsg.Recipients.Count Step 1
        strRecipName = objOneMsg.Recipients.Item(i).Name
        If strRecipName <> "" Then
            Set objOneRecip = objCopyMsg.Recipients.Add
            If objOneRecip Is Nothing Then
                MsgBox "unable to create recipient in message copy"
                Exit Function
            End If
            objOneRecip.Name = strRecipName
        End If
    Next i
```

See Also

Recipient Object

Send Method (Message Object)

Sends the message to the recipients via the MAPI system.

Syntax

object.**Send([***saveCopy, showDialog, parentWindow***])**

Parameters

object
Required. The Message object.

saveCopy
Optional. Boolean. If **True** or omitted, saves a copy of the Message in a user folder, such as the "Sent Messages" folder.

showDialog
Optional. Boolean. If **True**, displays a Send Message dialog box where the user can change the message contents or recipients.

parentWindow
Optional. Long. The parent window handle for the Send Message dialog box. A value of **0** (the default) specifies that any dialog box displayed is application modal. *parentWindow* is ignored unless *showDialog* is **True**.

Comments

The **Send** method is similar to the **Update** method, except **Send** ignores the parent folder object of the message and saves the message in the current user's default Outbox folder. Messaging systems retrieve messages from the Outbox and transport them to the recipients.

Note that the **Send** method invalidates the message object. Attempts to access the original message object result in an error. The original message object does not have to be set to **Nothing**, but it should not be used for subsequent operations. Use a new message object to obtain the message from the Outbox or from the Sent Messages folder.

The following methods can also invoke MAPI dialog boxes: **Delete** and **Details** methods (AddressEntry object), **Options** method (Message object), **Resolve** method (Recipient object and Recipients collection), **AddressBook** and **Logon** methods (Session object).

See Also

Creating and Sending a Message

Posting Messages to a Public Folder

Sent Property (Message Object)

Submitted Property (Message Object)

Sender Property (Message Object)

Returns the originator or original author of a message as an AddressEntry object. Read-only.

Syntax

Set *objAddrEntry* **=** *objMessage*.**Sender**

objAddrEntry
Object. The returned AddressEntry object that represents the message author.

objMessage
Object. The Message object.

Data Type

Object

Comments

The **Sender** property corresponds to the MAPI property PR_SENDER_ENTRYID.

Example

This example displays the name of the sender of a message:

```
' from the sample function Message_Sender
    Set objAddrEntry = objOneMsg.Sender
    If objAddrEntry Is Nothing Then
        MsgBox "Could not set the address entry object from the Sender"
        Exit Function
    End If
    MsgBox "Message was sent by " & objAddrEntry.Name
```

See Also

TimeReceived Property (Message Object)

Sent Property (Message Object)

True if the message has been sent through the MAPI system. Read-write.

Syntax

objMessage.**Sent**

Data Type

Boolean

Comments

In general, there are three different kinds of messages: messages that get *sent,* messages that get *posted,* and messages that get *saved*. Messages that get sent are characterized by traditional e-mail messages sent to a recipient or public folder. Messages that get posted are characterized by messages created in a public folder. Messages that get saved are characterized by messages that are created and saved without either sending or posting.

For all three kinds of messages, you use the **Submitted, Sent**, and **Unread** properties and the **Send** or **Update** methods.

The following table summarizes the use of the message properties and methods for these three kinds of messages:

Kind of message	Method	Submitted property	Sent property	Unread property
Gets sent	**Send**	**Send** method sets **True**	Spooler sets **True**	Spooler sets **True**
Gets posted	**Update**	Application sets **False**	Application sets **True**	Application sets **True**
Gets saved	**Update**	Application sets **False**	Application sets **False**	Application sets **True**

For messages that get sent, the **Sent** property can be written up until the time that you call the **Send** or **Update** method. Note that changing the **Sent** property to **True** does not cause the message to be sent. Only the **Send** method actually causes the message to be transmitted. After you call the **Send** method, the messaging system controls the **Sent** property and changes it to a read-only property.

A common use for writing a value to the **Sent** property is to set the property to **False** so that an electronic mail system can save pending, unsent messages in an Outbox folder, or to save work-in-progress messages in a Pending folder before committing the messages to a public information store. Note that you can cause an error if you set the property incorrectly.

The **Sent** property is changed using the following sequence. When you call the **Send** method to send a message to a recipient, the message is moved to the Outbox and the message object's **Submitted** property is set to **True**. When the messaging system spooler actually starts transporting the message, the **Sent** property is set to **True**.

When the message is not sent using the **Send** method, the MAPI system does not change the **Sent** property. For posted and saved messages that call the **Update** method, you should set the value of the **Sent** property to **True** just before you post the message.

The Sent **property corresponds to the MAPI property PR_MESSAGE_FLAGS and the flag MSGFLAG_UNSEND. See Also**

Creating and Sending a Message

Posting Messages to a Public Folder

Submitted Property (Message Object)

Signed Property (Message Object)

True if the message has been tagged with a digital signature. Read-write.

Syntax

objMessage.**Signed**

Data Type

Boolean

Comments

The **Signed** property is dependent upon the messaging or information store provider. OLE Messaging does not actually encrypt or digitally sign the message.

The **Signed** property corresponds to the MAPI property PR_SECURITY.

See Also

Encrypted Property (Message Object)

Securing Messages

Size Property (Message Object)

The approximate size in bytes of the message. Read-only.

Syntax

objMessage.**Size**

Data Type

Long

Comments

The **Size** property is not valid until after the first **Update** or **Send** operation.

The **Size** property corresponds to the MAPI property PR_MESSAGE_SIZE.

See Also

Attachments Property (Message Object)

Text Property (Message Object)

StoreID Property (Message Object)

Represents the unique identifier for the information store that contains this message. Read-only.

Syntax

objMessage.**StoreID**

Data Type

String

Comments

The **StoreID** property corresponds to the MAPI property PR_STORE_ENTRYID.

See Also

GetMessage Method (Session Object)

Sent Property (Message Object)

Subject Property (Message Object)

Returns or sets the subject of the message as a string. Read-write.

Syntax

objMessage.**Subject**

Data Type

String

Comments

The **Subject** property corresponds to the MAPI property PR_SUBJECT.

Example

This example sets the subject of a message:

```
Dim objMessage As Object  ' assume valid message
objMessage.Subject = "Microsoft Bob: Check It Out"
```

See Also

Text Property (Message Object)

Submitted Property (Message Object)

True when the message has been submitted. Read-write.

Syntax

objMessage.**Submitted**

Data Type

Boolean

Comments

In general, there are three different kinds of messages: messages that get *sent,* messages that get *posted,* and messages that get *saved*. Messages that get sent are characterized by traditional e-mail messages sent to a recipient or public folder. Messages that get posted are characterized by messages created in a public folder. Messages that get saved are characterized by messages that are created and saved without either sending or posting.

For all three kinds of messages, you use the **Submitted, Sent**, and **Unread** properties and the **Send** or **Update** methods.

The following table summarizes the use of the message properties and methods for these three kinds of messages:

Kind of message	Method	Submitted property	Sent property	Unread property
Gets sent	**Send**	**Send** method sets **True**	Spooler sets **True**	Spooler sets **True**
Gets posted	**Update**	Application sets **False**	Application sets **True**	Application sets **True**
Gets saved	**Update**	Application sets **False**	Application sets **False**	Application sets **True**

For messages that get sent, you create the message and then call the **Send** method. When the **Send** method is successful, the **Submitted** property is set to **True**. The value does not change after that point. For messages sent to a public folder, the **Send** method sets the **Submitted** property (rather than the **Sent** property) to **True**.

For messages that get posted, you create the message directly within a public folder and call **Update**. When you create the message within the public folder, some viewers do not allow the message to become visible to others until you set the **Submitted** property to **True**.

The **Submitted** property corresponds to the MAPI property PR_MESSAGE_FLAGS.

See Also

Send Method (Message Object)

Sent Property (Message Object)

Text Property (Message Object)

Returns or sets the body text of the message as a string. Read-write.

Syntax

object.**Text**

Data Type

String

Comments

The maximum size of the body text may be limited by the tool that you use to manipulate string variables (such as Microsoft Visual Basic).

Note that the **Text** property is a plain text representation of the message body and does not support rich text format (RTF).

The **Text** property corresponds to the MAPI property PR_BODY.

Example

This example sets the body text of a message:

```
Dim objMessage As Object    ' assume valid message
objMessage.Text = "Thank you for buying Microsoft Home(TM) products."
```

See Also

Subject Property (Message Object)

TimeReceived Property (Message Object)

Sets or returns the date and time the message was received as a **vbDate** variant data type. Read-write.

Syntax

object.**TimeReceived**

Data Type

Variant (**vbDate** format)

Comments

The **TimeReceived** and **TimeSent** properties set and return dates and times as the local time for the user's system.

When you send messages using the Message object **Send** method, the MAPI system sets the **TimeReceived** and **TimeSent** properties for you. However, when you post messages in a public folder, you must first explicitly set these properties. For a message posted to a public folder, set both properties to the same time value.

Note that the **TimeReceived** and **TimeSent** properties represent local time. However, when you access MAPI time properties through the Fields collection **Item** property, the time values represent Greenwich Mean Time (GMT).

The **TimeReceived** property corresponds to the MAPI Property PR_MESSAGE_DELIVERY_TIME.

Example

This example displays the date and time a message was sent and received:

```
' from the sample function Message_TimeSentAndReceived
   ' verify that objOneMsg is valid, then...
    With objOneMsg
        strMsg = "Message sent " & Format(.TimeSent, "Short Date")
        strMsg = strMsg & ", " & Format(.TimeSent, "Long Time")
        strMsg = strMsg & "; received "
        strMsg = strMsg & Format(.TimeReceived, "Short Date") & ", "
        strMsg = strMsg & Format(.TimeReceived, "Long Time")
        MsgBox strMsg
    End With
```

See Also

Item Property (Fields Collection)

TimeSent Property (Message Object)

TimeSent Property (Message Object)

Sets or returns the date and time the message was sent as a **vbDate** variant data type. Read-write.

Syntax

object.**TimeSent**

Data Type

Variant (**vbDate** format)

Comments

The **TimeReceived** and **TimeSent** properties return dates and times as the local time for the user's system.

When you send messages using the Message object **Send** method, the MAPI system sets the **TimeReceived** and **TimeSent** properties for you. However, when you post messages in a public folder, you must first explicitly set these properties. For a message posted to a public folder, set both properties to the same time value.

Note that the **TimeReceived** and **TimeSent** properties represent local time. However, when you access MAPI time properties through the Fields collection **Item** property, the time values represent Greenwich Mean Time (GMT).

The **TimeSent** property corresponds to the MAPI Property PR_CLIENT_SUBMIT_TIME.

Example

This example displays the date a message was sent and received:

```
' from the sample function Message_TimeSentAndReceived
   ' verify that objOneMsg is valid, then...
    With objOneMsg
        strMsg = "Message sent " & Format(.TimeSent, "Short Date")
        strMsg = strMsg & ", " & Format(.TimeSent, "Long Time")
        strMsg = strMsg & "; received "
        strMsg = strMsg & Format(.TimeReceived, "Short Date") & ", "
        strMsg = strMsg & Format(.TimeReceived, "Long Time")
        MsgBox strMsg
    End With
```

See Also

Item Property (Fields Collection)

TimeReceived Property (Message Object)

Type Property (Message Object)

Returns or sets the MAPI message class for the message. Read-write.

Syntax

object.**Type**

Data Type

String

Comments

The **Type** property returns or sets the MAPI message class for the message. By default, the OLE Messaging Library sets the **Type** value of new messages to the MAPI message class "IPM.Note."

The OLE Messaging Library does not impose any restrictions on this value except that it be a valid string value. You can set the value to any string that is meaningful for your application. MAPI uses message class strings in the form "IPM.*application.subClass*" or "IPC.*application.subClass*".

For more information about MAPI message classes, see the Microsoft *MAPI Programmer's Reference*.

The **Type** property corresponds to the MAPI property PR_MESSAGE_CLASS.

See Also

GetFirst Method (Messages Collection)

Unread Property (Message Object)

True if the message has not been read by the current user. Read-write.

Syntax

object.**Unread**

Data Type

Boolean

Comments

When you post a message to a public folder, you should set the **Unread**, **Submitted**, and **Sent** properties to **True** before calling the **Send** or **Update** method.

The Unread **property corresponds to the MAPI property PR_MESSAGE_FLAGS.See Also**

Posting a Message to a Public Folder

Sent Property (Message Object)

Update Method (Message Object)

Saves the message in the MAPI system.

Syntax

objMessage.**Update()**

Parameters

objMessage
Required. The Message object.

Comments

Changes to Message objects are not permanently saved in the MAPI system until you use the **Update** method. This example changes the subject of the first message in a folder:

```
Set objMessage = objSession.Inbox.GetFirst
' ... verify message
objMessage.Subject = "This is the new subject"
objMessage.Update
```

To add a new Message object, use the **Add** method followed by the **Update** method. Both of these examples save a new message:

```
Dim objMessage As Object   ' message object
' ...
Set objMessage = objSession.Outbox.Messages.Add
objMessage.Subject = "Microsoft Bob(TM)"
objMessage.Text = "This is incredible, you've got to see it!"
objMessage.Update
```

See Also

Send Method (Message Object)

Messages Collection Object

The Messages collection object contains one or more Message objects.

The Messages collection is considered a *large collection*, which means that you must use a Message ID value or the **Get*** methods to access individual Message objects within the collection.

Properties

Property name	Type	Access
Application	String	Read-only
Class	Long	Read-only
Parent	Object	Read-only
Session	Session object	Read-only

Methods

Method name	Parameters
Add	(optional) subject as String, (optional) text as String, (optional) type as String, (optional) importance as Long
Delete	(none)
GetFirst	(optional) filter as Variant (Note: Version 1.0 supports strings only)
GetLast	(optional) filter as Variant (Note: Version 1.0 supports strings only)
GetNext	(optional) filter as Variant (Note: Version 1.0 supports strings only)
GetPrevious	(optional) filter as Variant (Note: Version 1.0 supports strings only)
Sort	sortOrder as Long

Comments

The collection does not maintain a count of the number of Message objects in the collection. Use the **GetFirst, GetLast, GetNext, and GetPrevious** methods to access the Message objects in the collection.

The order in which items are returned by **GetFirst, GetNext**, **GetPrevious**, and **GetLast** depends on whether the messages are sorted or not. The Message objects within a collection can be sorted by delivery time, either ascending or descending.

When the items are unsorted, these methods do not return the items in any specified order. The best programming approach to use with unsorted collections is to assume that you can access all items within the collection, but that the order of the objects is undefined.

See Also

OLE Messaging Object Collections

Add Method (Messages Collection)

Creates and returns a new Message object in the Messages collection.

Syntax

Set *objMessage* = *objMsgCollection*.**Add([***subject, text, type, importance***])**

Parameters

objMessage
On successful return, represents the new Message object added to the collection.

objMsgCollection
Required. The Messages collection object.

subject
Optional. String. The subject of the message.

text
Optional. String. The body text of the message.

type
Optional. String. The message class of the message, such as the default, "IPM.Note."

importance
Optional. Long. The priority of the message. The following values are defined:

Constant	Value	Description
mapiLow	0	Low priority
mapiNormal	1	Normal priority (default)
mapiHigh	2	High priority

Comments

The method parameters correspond to the **Subject, Text, Type,** and **Importance** properties of the Message object.

You should create new messages in the Outbox folder.

Example

This example adds a new message to a folder:

```
' from the sample function Util_ReplyToConversation
    Set objNewMsg = objSession.Outbox.Messages.Add
    ' verify objNewMsg created successfully...then supply properties
    With objNewMsg
        .Text = "How about a slightly used Gemini?"   ' new text
        .Subject = objOriginalMsg.Subject   ' copy original properties
        .ConversationTopic = objOriginalMsg.ConversationTopic
        ' append time stamp; compatible with Microsoft Exchange client
        Set objOneRecip = .Recipients.Add( _
                    Name:=objOriginalMsg.Recipients.Item(1).Name, _
                    Type:=mapiTo)
        .Recipients.Resolve
        .Update
        .Send showDialog:=False
    End With
```

See Also

Delete Method (Message Object)

Delete Method (Messages Collection)

Deletes all messages in the collection.

Syntax

object.**Delete()**

Parameters

object
Required. The Messages collection object.

Comments

Deletes all member objects within the collection.

Moves the deleted folders and messages to the Deleted Messages folder, if the user has enabled this option. If the folders and messages are already in the Deleted Messages folder, the **Delete** method permanently deletes them, and they cannot be recovered.

Be cautious using **Delete** with collections, since the method deletes all member objects within a collection. For example, this code deletes all of the messages in a folder:

```
objFolder.Messages.Delete
```

See Also

Add Method (Messages Collection)

GetFirst Method (Messages Collection)

Returns the first object in the collection. Returns **Nothing** if no first object exists.

Syntax

Set *objMessage* **=** *objMsgColl*.**GetFirst([***filter***])**

Parameters

objMessage
On successful return, represents the first Message object in the collection.

objMsgColl
Required. The Messages collection object.

filter
Optional. Version 1.0 supports a string that specifies the message class of the object, such as "IPM.Note." Corresponds to the **Type** property of the Message object.

Comments

The **Get*** methods are similar to the **Find*** and **Move*** methods that are used with Microsoft Access databases. The **Get*** methods take a different name from these methods because they use a different syntax.

See Also

Message Object

GetLast Method (Messages Collection)

Returns the last object in the collection. Returns **Nothing** if no last object exists.

Syntax

Set *objMessage* **=** *objMsgColl*.**GetLast([***filter***])**

Parameters

objMessage
On successful return, represents the last Message object in the collection.

objMsgColl
Required. The Messages collection object.

filter
Optional. Version 1.0 supports a string that specifies the message class of the object, such as "IPM.Note." Corresponds to the **Type** property of the Message object.

Comments

The **Get*** methods are similar to the **Find*** and **Move*** methods that are used with Microsoft Access databases. The **Get*** methods take a different name from these methods because they use a different syntax.

See Also

Update Method (Message Object)

GetNext Method (Messages Collection)

Returns the next object in the Messages collection. Returns **Nothing** if no next object exists.

Syntax

Set *objMessage* **=** *objMsgColl*.**GetNext([***filter***])**

Parameters

objMessage
On successful return, represents the next Message object in the collection.

objMsgColl
Required. The Messages collection object.

filter
Optional. Version 1.0 supports a string that specifies the message class of the object, such as "IPM.Note." Corresponds to the **Type** property of the Message object.

Comments

The **Get*** methods are similar to the **Find*** and **Move*** methods that are used with Microsoft Access databases. The **Get*** methods take a different name from these methods because they use a different syntax.

See Also

Update Method (Message Object)

GetPrevious Method (Messages Collection)

Returns the previous object in the collection. Returns **Nothing** if no previous object exists.

Syntax

Set *objMessage* **=** *objMsgColl*.**GetPrevious([***filter***])**

Parameters

objMessage
On successful return, represents the previous Message object in the collection.

objMsgColl
Required. The Messages collection object.

filter
Optional. Version 1.0 supports a string that specifies the message class of the object, such as "IPM.Note." Corresponds to the **Type** property of the Message object.

Comments

The **Get*** methods are similar to the **Find*** and **Move*** methods that are used with Microsoft Access databases. The **Get*** methods take a different name from these methods because they use a different syntax.

See Also

Update Method (Message Object)

Sort Method (Messages Collection)

Sorts the messages in the collection according to the specified sort order.

Syntax

objMsgColl.**Sort(***sortOrder***)**

Parameters

objMsgColl
Required. The Messages collection object.

sortOrder
Required. Long. The specified sort order, one of the following values:

Value	Numeric Value	Description
mapiNone	0	No sort
mapiAscending	1	Ascending sort
mapiDescending	2	Descending sort

See Also

Update Method (Message Object)

Recipient Object

Represents a recipient of a message.

Properties

Property name	Type	Access
Address	String	Read-write
AddressEntry	AddressEntry object	Read-write
Application	String	Read-only
Class	Long	Read-only
Index	Long	Read-only
Name	String	Read-write
Parent	Object	Read-only
Session	Session object	Read-only
Type	Long	Read-write

Methods

Method name	Parameters
Delete	(none)
Resolve	(optional) showDialog as Boolean

See Also

AddressEntry Object

Recipients Collection

Address Property (Recipient Object)

Specifies the *full address* for this recipient. Read-write.

Syntax

object.**Address**

Data Type

String

Comments

Set the value of the Recipient object's **Address** property to specify a custom address. The Recipient **Address** uses the following syntax:

TypeValue:*AddressValue*

where *TypeValue* and *AddressValue* correspond to the values of the AddressEntry object's **Type** and **Address** properties.

The Recipient object's **Address** property represents the *full address*, the complete messaging address used by the MAPI system.

OLE Messaging version 1.0 sets the value of the Recipient object's **Address** property for you when you supply the **Name** property and call its **Resolve** method.

The **Address** property corresponds to the MAPI properties PR_EMAIL_ADDRESS and PR_EMAIL_TYPE.

Example

```
' from the sample function Util_CompareAddressParts
    strMsg = "Recipient full address = " & objOneRecip.Address
    strMsg = strMsg & "; AddressEntry type = " & objAddrEntry.Type
    strMsg = strMsg & "; AddressEntry address = " & objAddrEntry.Address
    MsgBox strMsg    ' compare address components
```

See Also

AddressEntry Object

Name Property (Recipient Object)

Resolve Method (Recipient Object)

AddressEntry Property (Recipient Object)

Specifies the AddressEntry object for this recipient. Read-write.

Syntax

objRecipient.**AddressEntry**

Data Type

Object (AddressEntry object)

Comments

The **AddressEntry** property indicates the AddressEntry object for this recipient. For example, you can use a Recipient object's child AddressEntry object or the Message object's **Sender** property.

Example

```
' from the sample function Session_AddressEntry
    If objOneRecip Is Nothing Then
        MsgBox "must select a recipient"
        Exit Function
    End If
    Set objAddrEntry = objOneRecip.AddressEntry
    If objAddrEntry Is Nothing Then
        MsgBox "no valid AddressEntry for this recipient"
        Exit Function
    End If
' from the sample function Util_CompareAddressParts
    strMsg = "Recipient full address = " & objOneRecip.Address
    strMsg = strMsg & "; AddressEntry type = " & objAddrEntry.Type
    strMsg = strMsg & "; AddressEntry address = " & objAddrEntry.Address
    MsgBox strMsg    ' compare address components
    strMsg = "Recipient name = " & objOneRecip.Name
    strMsg = strMsg & "; AddressEntry name = " & objAddrEntry.Name
    MsgBox strMsg    ' compare display names (should be same)
```

See Also

AddressEntry Object

Delete Method (Recipient Object)

Deletes the Recipient object.

Syntax

object.**Delete()**

Parameters

object
Required. The Recipient object.

Comments

The object is set to **Nothing** and it is removed from memory. The change is not permanent until you use the **Update**, **Send**, or **Delete** method on the parent Message object.

See Also

Send Method (Message Object)

Update Method (Message Object)

Index Property (Recipient Object)

Returns the index number of this Recipient object within the Recipients collection. Read-only.

Syntax

objRecipient.**Index**

Data Type

Long

Comments

The index number indicates an index within the array of Recipients collection object.

Example

```
' from the sample function Recipients_NextItem
'   after some similar validation...
    If iRecipCollIndex >= objRecipColl.Count Then
        iRecipCollIndex = objRecipColl.Count
        MsgBox "Already at end of recipient list"
        Exit Function
    End If
    ' index is < count; can be incremented by 1
    iRecipCollIndex = iRecipCollIndex + 1
    Set objOneRecip = objRecipColl.Item(iRecipCollIndex)
' from the sample function Recipient_Index
    If objOneRecip Is Nothing Then
        MsgBox "must first select a recipient"
        Exit Function
    End If
    MsgBox "Recipient index = " & objOneRecip.Index
```

See Also

Count Property (Recipients Collection)

Item Property (Recipients Collection)

Name Property (Recipient Object)

Returns the name of this Recipient object. Read-write.

Syntax

object.**Name**

Data Type

String

Comments

The **Name** property corresponds to the MAPI property PR_DISPLAY_NAME.

Example

```
' from the sample function Util_CompareFullAddressParts()
Dim strMsg As String
    ' validate objects... then display
    strMsg = "Recipient full address = " & objOneRecip.Address
    strMsg = strMsg & "; AddressEntry type = " & objAddrEntry.Type
    strMsg = strMsg & "; AddressEntry address = " & objAddrEntry.Address
    MsgBox strMsg    ' compare address parts

    strMsg = "Recipient name = " & objOneRecip.Name
    strMsg = strMsg & "; AddressEntry name = " & objAddrEntry.Name
    MsgBox strMsg    ' compare display names (should be same)
```

See Also

Recipient Object

Resolve Method (Recipient Object)

Resolves address information. When the Recipient **Name** property is supplied, looks up the corresponding address from the Address Book. When the Recipient **Address** property is supplied, resolves as a custom address.

Syntax

object.**Resolve([** *showDialog* **])**

Parameters

object
Required. The Recipient object.

showDialog
Optional. Boolean. If **True**, displays a dialog box to prompt the user to resolve ambiguous names.

Comments

The **Resolve** method operates when the **AddressEntry** property is set to **Nothing**. Its operation depends on whether you supply the Recipient **Name** or **Address** property.

When you supply the **Name** property, **Resolve** looks up the Recipient object's **Name** property in the Address Book. When a recipient is resolved, the recipient object's **Address** property contains the full address and its **AddressEntry** property contains a reference to an AddressEntry object that represents a copy of information in the address book.

When you specify a custom address by supplying the Recipient's **Address** property, the **Resolve** method does not attempt to compare the address against the Address Book.

To avoid delivery errors, clients should always resolve recipients before submitting a message to the MAPI system. Resolving the recipient name means either finding a matching address in an address list or having the user select an address from a dialog.

The **Resolve** method uses the address list specified in the profile, such as the Global Address Book or the Personal Address Book.

The following methods can also invoke MAPI dialogs: **Delete** and **Details** methods (AddressEntry object), **Options** and **Send** methods (Message object), **Resolve** method (Recipient object and Recipients collection), **AddressBook** and **Logon** methods (Session object).

See Also

Resolve Method (Recipients Collection)

Resolved Property (Recipients Collection)

Type Property (Recipient Object)

Specifies the type of the Recipient object; an integer value that indicates either "To:", "Cc:", or "Bcc:". Read-write.

Syntax

objRecipient.**Type**

Data Type

Integer

Comments

The **Type** property applies to all Recipient objects in the Recipients collection. The property has the following defined values:

Recipient type	Value	Description
mapiTo	1	The recipient is on the To: line.
mapiCc	2	The recipient is on the Cc: line.
mapiBcc	3	The recipient is on the Bcc: line.

The **Type** property corresponds to the MAPI property PR_RECIPIENT_TYPE.

See Also

Address Property (Recipient Object)

Resolve Method (Recipient Object)

Recipients Collection Object

The Recipients collection object specifies the recipients of a message.

The Recipients collection is considered a *small collection*, which means that it supports count and index values that let you access individual Recipient objects through the **Item** property.

Properties

Property name	Type	Access
Application	String	Read-only
Class	Long	Read-only
Count	Long	Read-only
Item	Recipient object	Read-only
Parent	Object	Read-only
Resolved	Boolean	Read-only
Session	Session object	Read-only

Methods

Method name	Parameters
Add	(optional) name as String, (optional) address as String, (optional) type as Long, (optional) entryID as String
Delete	(none)
Resolve	(optional) showDialog as Boolean

See Also

OLE Messaging Object Collections

Add Method (Recipients Collection)

Creates a new Recipient object in the Recipients collection.

Syntax

Set *objRecipient* = *objRecipColl*.**Add([***name, address, type, entryID***])**

Parameters

objRecipient
On successful return, represents the new Recipient object added to the collection.

objRecipColl
Required. The Recipients collection object.

name
Optional. String. The display name of the recipient.

address
Optional. String. The address of the recipient.

type
Optional. Long. The type of recipient.

entryID
Optional. String. The ID of a valid AddressEntry object for this recipient.

Comments

The *name*, *address*, and *type* parameters correspond to the Recipient object **Name**, **Address**, and **Type** properties, respectively. The *entryID* parameter corresponds to the child AddressEntry object's **ID** property. You can access the child AddressEntry object through the Recipient object's **AddressEntry** property.

Call the **Resolve** method after you add a recipient.

The **Index** of the new Recipient object equals the new **Count** of the Recipients collection. The recipient is actually saved in the MAPI system when you **Update** or **Send** the parent message object.

Example

This example adds three recipients to a message. The address for the first recipient is resolved using the display name. The second recipient is a custom address, so the resolve operation does not modify it. The third recipient is taken from an existing valid AddressEntry object. The Resolve operation does not affect this recipient.

```
' from the sample function "Using Addresses"

    ' add 3 recipient objects to a valid message object
    ' 1. look up entry in address book
    Set objOneRecip = objNewMessage.Recipients.Add( _
        Name:=strName, _
        Type:=mapiTo)
    ' error handling...verify objOneRecip
    objOneRecip.Resolve

    ' 2. add a custom recipient
    Set objOneRecip = objNewMessage.Recipients.Add( _
        Address:="SMTP:davidhef@microsoft.com", _
```

```
    Type:=mapiTo)
If objOneRecip Is Nothing Then
    MsgBox "Unable to add recipient using custom addressing"
    Exit Function
End If
objOneRecip.Resolve

' 3. add a valid address entry object, such as Message.Sender
' assume valid address entry ID, name from an existing message
Set objOneRecip = objNewMessage.Recipients.Add( _
    entryID:=strAddrEntryID, _
    Name:=strName, _
    Type:=mapiTo)
If objOneRecip Is Nothing Then
    MsgBox "Unable to add recipient using existing AddressEntry ID"
    Exit Function
End If

objNewMessage.Text = "expect 3 different recipients"
MsgBox ("count = " & objNewMessage.Recipients.Count)
```

See Also

Resolve Method (Recipients Collection)

Count Property (Recipients Collection)

Returns the number of Recipient objects in the collection. Read-only.

Syntax

object.**Count**

Data Type

Long

Example

This example lists the names of all recipients in the collection:

```
' from the sample function Util_CopyMessage
' Copy all Recipient objects from one collection to another
'  ... verify valid message object objOneMsg
   For i = 1 To objOneMsg.Recipients.Count Step 1
        strRecipName = objOneMsg.Recipients.Item(i).Name
        If strRecipName <> "" Then
            Set objOneRecip = objCopyMsg.Recipients.Add
            If objOneRecip Is Nothing Then
                MsgBox "unable to create recipient in message copy"
                Exit Function
            End If
            objOneRecip.Name = strRecipName
        End If
    Next i
```

See Also

Item Property (Recipients Collection)

Delete Method (Recipients Collection)

Deletes all Recipients in the collection.

Syntax

object.**Delete()**

Parameters

object
Required. The Recipients collection object.

Comments

The object or collection is set to **Nothing** and it is removed from memory, but the change is not permanent until you use the **Update**, **Send**, or **Delete** method on the parent object that contained the deleted object or collection.

Be cautious using **Delete** with collections, since the method deletes all member objects within a collection.

See Also

Delete Method (Recipient Object)

Send Method (Message Object)

Update Method (Message Object)

Item Property (Recipients Collection)

Returns a single Recipient from the collection. Read-only.

Syntax

objRecipCollection.**Item(***index***)**

objRecipCollection
Required. Specifies the Recipients collection object.

index
An integer that ranges from 1 to *objRecipCollection*.**Count**, or a string that specifies the name of the object.

Data Type

Object

Comments

The **Item** property works like the accessor property for a collection.

Example

```
' fragment from Util_ReplyToConversation
'  use an index to access each entry in the Recipients collection
'  1 represents the first item, Recipients.Count represents the last
Set objOneRecip = .Recipients.Add( _
                      Name:=objOriginalMsg.Recipients.Item(1).Name, _
                      Type:=mapiTo)
```

See Also

Count Property (Recipients Collection)

Resolve Method (Recipients Collection)

Searches the Recipients collection to resolve names.

Syntax

objRecipColl.**Resolve([** *showDialog* **])**

Parameters

objRecipColl
Required. The Recipients collection object.

showDialog
Optional. Boolean. If **True**, displays a dialog box to prompt the user to resolve ambiguous names.

Comments

Calling the Recipients collection **Resolve** method is equivalent to calling the **Resolve** method for each recipient object in the collection. For more information about the individual Resolve operation, see the reference documentation for the Recipient object **Resolve** method.

The following methods can also invoke MAPI dialog boxes: **Delete** and **Details** methods (AddressEntry object), **Options** and **Send** methods (Message object), **Resolve** method (Recipient object), **AddressBook** and **Logon** methods (Session object).

Example

```
' from the sample function Util_NewConversation
'   create a valid new message object in the Outbox
    With objNewMsg
        .Subject = "used space vehicle wanted"
        ' ... set other properties here...
        Set objOneRecip = .Recipients.Add(Name:="Car Ads", Type:=mapiTo)
        If objOneRecip Is Nothing Then
            MsgBox "Unable to create the public folder recipient"
            Exit Function
        End If
        .Recipients.Resolve
    End With
```

See Also

Resolve Method (Recipient Object)

Resolved Property (Recipients Collection)

Resolved Property (Recipients Collection)

True if all of the recipients in the collection are resolved. Read-only.

Syntax

object.**Resolved**

Data Type

Boolean

Comments

All recipient objects in the collection are considered resolved when all Recipient objects have a valid address entry object in the **AddressEntry** property.

You should resolve all addresses. Whenever you supply a display name to obtain an address from the address book or supply a custom address, you should call the **Resolve** method to ensure that the **AddressEntry** property is valid.

When the Recipients collection **Resolved** property is not **True**, use either the collection's **Resolve** method or the **Resolve** method for each recipient object in the collection.

When you use existing valid AddressEntry objects, you do not need to explicitly call the Recipient object's **Resolve** method.

See Also

Resolve Method (Recipient Object)

Session Object

The Session object contains session-wide settings and options. It also contains properties that return top-level objects, such as **CurrentUser**.

Properties

Property name	Type	Access
Application	String	Read-only
Class	Long	Read-only
CurrentUser	AddressEntry object	Read-only
Inbox	Folder object	Read-only
MAPIOBJECT	Object	Read-write (Note: Not available to Visual Basic applications)
Name	String	Read-only
OperatingSystem	String	Read-only
Outbox	Folder object	Read-only
Parent	Object; set to **Nothing**	(Not applicable)
Session	Object; set to **Nothing**	(Not applicable)
Version	String	Read-only

Methods

Method name	Parameters
AddressBook	(optional) recipients as Object, (optional) title as String, (optional) oneAddress as Boolean, (optional) forceResolution as Boolean, (optional) recipLists as long, (optional) toLabel as String, (optional) ccLabel as String, (optional) bccLabel as String, (optional) parentWindow as Long
GetAddressEntry	entryID as String
GetFolder	folderID as String, storeID as String
GetMessage	messageID as String, storeID as String
Logoff	(none)
Logon	(optional) profileName as String, (optional) profilePassword as String, (optional) showDialog as Boolean, (optional) newSession as Boolean, (optional) parentWindow as Long

Comments

After you create a new Session object, use the **Logon** method to initiate a MAPI session.

See Also

Logon Method (Session Object)

AddressBook Method (Session Object)

Displays the MAPI dialog box that allows the user to select entries from the Address Book. The selections are returned in a Recipients collection object.

Syntax

Set *objRecipients* **=** *objSession*.**AddressBook(** [*recipients*, *title, oneAddress, forceResolution, recipLists, toLabel, ccLabel, bccLabel, parentWindow*] **)**

Parameters

objRecipients
On successful return, the Recipients collection object. When the user does not select any names from the dialog box, AddressBook returns **Nothing**.

objSession
Required. The Session object.

recipients
Optional. Object. A Recipients collection object that provides the initial value for the recipient list boxes in the Address Book. (Note: This initial Recipient collection is ignored in OLE Messaging Library version 1.0.)

title
Optional. String. The title or caption of the Address Book dialog box.

oneAddress
Optional. Boolean. Allows the user to enter or select only one address.

forceResolution
Optional. Boolean. If **True**, attempts to resolve all names before closing the AddressBook. Prompts the user to resolve any ambiguous names.

recipLists
Optional. Long. The number of recipient list boxes to display in the Address Book dialog:

recipLists	Action
0	Displays no list boxes. The user can select one or more names, but the user cannot enter custom names.
1	Displays 1 list box (default).
2	Displays 2 list boxes.
3	Displays 3 list boxes.

toLabel
Optional. String. The caption for the first list box. Ignored if *recipLists* is less than 1. If omitted, the default value "To:" is displayed.

ccLabel
Optional. String. The caption for the second list box. Ignored if *recipLists* is less than 2. If omitted, the default value "CC:" is displayed.

bccLabel
Optional. String. The caption for the third list box. Ignored if *recipLists* is less than 3. If omitted, the default value "BCC:" is displayed.

parentWindow
Optional. Long. The parent window handle for the Address Book dialog box. A value of **0** (the default) specifies that any dialog box displayed is application modal. *parentWindow* is ignored unless *showDialog* is **True**.

Comments

The **AddressBook** method returns **Nothing** if the user cancels the dialog box.

To provide an access key for the list boxes, include an ampersand ("&") character in the string for the label argument. For example, if *toLabel* is "&Attendees:", users can press ALT+A to move the focus to the first recipient list box.

The following methods can also invoke MAPI dialog boxes: **Delete** and **Details** methods (AddressEntry object), **Options** and **Send** methods (Message object), **Resolve** method (Recipient object and Recipients collection), **Logon** method (Session object).

Note The initial Recipient collection, as specified in the *recipients* parameter, is not used in OLE Messaging Library version 1.0.

Example

The following example displays an address book with one recipient list labelled "Attendees":

```
Function Session_AddressBook()
    On Error GoTo err_Session_AddressBook

    If objSession Is Nothing Then
        MsgBox "must first create MAPI session and logon"
        Exit Function
    End If
    Set objRecipColl = objSession.AddressBook( _
        Title:="Select Attendees", _
        forceResolution:=True, _
        recipLists:=1, _
        toLabel:="&OLE Messaging")  ' appears on button
    ' Note: initial value not used in version 1.0
    ' parameter not used in call: Recipients:=objInitRecipColl
    MsgBox "Name of first recipient = " & objRecipColl.Item(1).Name
    Exit Function

err_Session_AddressBook:
    If (Err = 91) Then   ' object not set
        MsgBox "No recipients selected"
    Else
        MsgBox "Unrecoverable Error:" & Err
    End If
    Exit Function
End Function
```

See Also

AddressEntry Object

Recipients Collection

CurrentUser Property (Session Object)

Returns the active user as an AddressEntry object. Read-only.

Syntax

objSession.**CurrentUser**

Data Type

Object (AddressEntry object)

Comments

The **CurrentUser** property returns **Nothing** when no user is logged on.

Example

The example logs on if necessary, then creates strings containing information about the current user:

```
If objSession Is Nothing Then
    MsgBox ("Must log on first")
    Exit Function
End If
Set objAddrEntry = objSession.CurrentUser
If objAddrEntry Is Nothing Then
    MsgBox "Could not set the address entry object"
    Exit Function
Else
    MsgBox "full address = " & objAddrEntry.Type & ":" _
             & objAddrEntry.Address
End If
```

See Also

AddressEntry Object

GetAddressEntry Method (Session Object)

Returns an AddressEntry object.

Syntax

Set *objAddressEntry* **=** *objSession*.**GetAddressEntry(***entryID***)**

Parameters

objAddressEntry
On successful return, represents the AddressEntry object specified by *entryID*.

objSession
Required. The Session object.

entryID
Required. String that specifies the unique ID of the address entry.

Example

The following example displays the name of a user from a MAPI address list:

```
' from the function Session_GetAddressEntry
    If objSession Is Nothing Then
        MsgBox "No active session, must log on"
        Exit Function
    End If
    If "" = strAddressEntryID Then
        MsgBox ("Must first set string variable; see AddressEntry->ID")
        Exit Function
    End If
    Set objAddrEntry = objSession.GetAddressEntry(strAddressEntryID)
    MsgBox "full address = " & objAddrEntry.Type & ":" _
            & objAddrEntry.Address
```

See Also

Using Addresses

AddressEntry Object

GetFolder Method (Session Object)

GetMessage Method (Session Object)

GetFolder Method (Session Object)

Returns a folder object from a MAPI information store.

Syntax

Set *objFolder* = *objSession*.**GetFolder(***folderID* **[,** *storeID***])**

Parameters

objFolder
On successful return, contains the Folder object with the specified ID. When the Folder does not exist, **GetFolder** returns **Nothing**.

objSession
Required. The Session object.

folderID
Required. String that specifies the unique ID of the folder.

storeID
Optional. String that specifies the unique ID of the store.

Comments

The **GetFolder** method allows you to obtain any folder for which you know the ID.

Example

The following example uses the **GetFolder** method to obtain a specific folder from a MAPI information store:

```
' from the function Session_GetFolder
' requires a global variable that contains the folder ID
' uses a global variable that contains the store ID if present
    If strFolderID = "" Then
        MsgBox ("Must first set folder ID variable; see Folder->ID")
        Exit Function
    End If
    If strFolderStoreID = "" Then ' if it's not there, don't use it
        Set objFolder = objSession.GetFolder(strFolderID)
    Else
        Set objFolder = objSession.GetFolder(folderID:=strFolderID, _
                                    storeID:=strFolderStoreID)
    End If
    If objFolder Is Nothing Then
        Set objMessages = Nothing
        MsgBox "Unable to retrieve folder with specified ID"
        Exit Function
    End If
    MsgBox "Folder set to " & objFolder.Name
    Set objMessages = objFolder.Messages
```

See Also

Folder Object

ID Property (Folder Object)

GetMessage Method (Session Object)

Returns a Message object from a MAPI information store.

Syntax

Set *objMessage* = *objSession*.**GetMessage(***messageID* **[,** *storeID***])**

Parameters

objMessage
On successful return, **GetMessage** returns a Message object. When the specified *messageID* does not exist, **GetMessage** returns **Nothing**.

objSession
Required. The Session object.

messageID
Required. String that specifies the unique ID of the message.

storeID
Optional. String that specifies the unique ID of the store.

Example

The following example displays the subject of a message from a MAPI information store:

```
' fragment from Session_GetMessage
' requires the parameter strMessageID;
' also uses strMessageStoreID if it is defined
    If strMessageID = "" Then
        MsgBox ("Must first set message ID variable; see Message->ID")
        Exit Function
    End If
    If strMessageStoreID = "" Then ' not present
        Set objOneMsg = objSession.GetMessage(strMessageID)
    Else
        Set objOneMsg = objSession.GetMessage(messageID:=strMessageID, _
                                    storeID:=strMessageStoreID)
    End If
```

See Also

ID Property (Message Object)

Message Object

Inbox Property (Session Object)

Returns a Folder object representing the current user's default Inbox folder.

Syntax

objSession.**Inbox**

Data Type

Object (Folder object)

Comments

These properties return **Nothing** when the current user does not have or has not enabled these folders.

In addition to the formal collection and object hierarchy, OLE Messaging supports a simpler relationship between the Session object and certain common folders. This simplified structure more readily supports basic messaging-enabled applications. If the current user's default Inbox folder is named "Inbox," then these two folders are equivalent:

```
' from the function Session_Inbox
    '   make sure the Session object is valid...
    Set objFolder = objSession.Inbox
    If objFolder Is Nothing Then
        MsgBox "Failed to open Inbox"
        Exit Function
    End If
    MsgBox "Folder name = " & objFolder.Name
    Set objMessages = objFolder.Messages
    If objMessages Is Nothing Then
        MsgBox "Failed to open folder's Messages collection"
        Exit Function
    End If
```

See Also

Folder Object

Outbox Property (Session Object)

Logoff Method (Session Object)

Logs off from the MAPI system.

Syntax

object.**Logoff()**

Parameters

object
Required. The Session object.

Example

The following example logs off from the MAPI system:

```
' from the function Session_Logoff
    If Not objSession Is Nothing Then
        objSession.Logoff
        MsgBox "Logged off; reset global variables"
    Else
        MsgBox "No active session"
    End If
```

See Also

Logon Method (Session Object)

Logon Method (Session Object)

Logs on to the MAPI system.

Syntax

object.**Logon(** [*profileName, profilePassword, showDialog, newSession, parentWindow*] **)**

Parameters

object
Required. The Session object.

profileName
Optional. A string specifying the user's logon name. To prompt the user to enter a logon name, omit *profileName* and set *showDialog* to **True**.

profilePassword
Optional. A string specifying the user's logon password. To prompt the user to enter a logon password, omit *profilePassword* and set *showDialog* to **True**.

showDialog
Optional. Boolean. If **True**, displays a logon dialog box.

newSession
Optional. Boolean. Determines whether the application opens a new MAPI session or uses the current shared MAPI session. If a shared MAPI session does not exist, *newSession* is ignored and a new session is opened. If the shared MAPI session does exist, this argument takes the following action:

Value	Action
True	Opens an new MAPI session.
False or omitted	Uses the current, shared MAPI session.

parentWindow
Optional. Long (HWND). Specifies the parent window handle for the logon dialog box. A value of **0** (the default) specifies that any dialog box displayed is application modal. *parentWindow* is ignored unless *showDialog* is **True**.

Comments

The user must log on before your application can use most MAPI objects.

The following methods can also invoke MAPI dialog boxes: **Delete** and **Details** methods (AddressEntry object), **Options** and **Send** methods (Message object), **Resolve** method (Recipient object and Recipients collection), **AddressBook** method (Session object).

Example

The first example displays a logon dialog box that prompts the user to enter a logon password. The second example supplies the profilename parameter and does not display the dialog:

```
' from the function Session_Logon
    Set objSession = CreateObject("MAPI.Session")
    If Not objSession Is Nothing Then
        objSession.Logon showDialog:=True
    End If

' from the function Session_Logon_NoDialog
```

```
Function Session_Logon_NoDialog()
    On Error GoTo error_olemsg
    ' can set strProfileName, strPassword from a custom form
    ' adjust these parameters for your configuration
    ' create a Session object if necessary here...
    '
    If Not objSession Is Nothing Then
        ' configure these parameters for your needs...
        objSession.Logon profileName:=strProfileName, showDialog:=False
    End If
    Exit Function

error_olemsg:
    If 1273 = Err Then
        MsgBox "cannot logon: incorrect profile name or password; change
global variable strProfileName in Util_Initialize"
        Exit Function
    End If
    MsgBox "Error " & Str(Err) & ": " & Error$(Err)
    Resume Next
End Function
```

See Also

Starting A Session With MAPI

Logoff Method (Session Object)

MAPIOBJECT Property (Session Object)

Returns an **IUnknown** pointer to this Folder object. Not available to Visual Basic applications.

Syntax

object.**MAPIOBJECT**

Data Type

Variant (VT_UNKNOWN)

Comments

The **MAPIOBJECT** property is not available to Visual Basic programs. It is available only to C/C++ programs that use the OLE Messaging Library. The **MAPIOBJECT** property is an **IUnknown** object, which is not supported by Visual Basic. Visual Basic supports **IDispatch** objects. For more information, see the Microsoft *OLE 2 Programmer's Reference*.

See Also

A Short Tour of OLE Automation

Name Property (Session Object)

Returns the name of the profile logged on to this session. Read-only.

Syntax

objSession.**Name**

Data Type

String

Comments

The **Name** property corresponds to the MAPI property PR_DISPLAY_NAME.

Examples

```
' from the function Session_Name
    If objSession Is Nothing Then
        MsgBox "Must log on first: see Session menu"
        Exit Function
    End If
    MsgBox "Session name = " & objSession.Name
```

See Also

Session Object

OperatingSystem Property (Session Object)

Returns the name and version number of the current operating system. Read-only.

Syntax

objSession.**OperatingSystem**

Data Type

String

Comments

OLE Messaging returns strings in the following formats:

Operating system	String value
Microsoft Windows NT	Microsoft® Windows NT™ *x.xx*
Microsoft Windows for Workgroups	Microsoft® Windows™ *x.xx*

The *x.xx* values are replaced with the actual version numbers. Note that Microsoft Windows for Workgroups version 3.11 returns the string "Microsoft® Windows™ 3.10." This is a feature of that operating system rather than a feature of the OLE Messaging Library.

Example

This example displays the name of the operating system:

```
' from the function Session_OperatingSystem
    If objSession Is Nothing Then
        MsgBox "Must log on first: see Session menu"
        Exit Function
    End If
    MsgBox "Operating system = " & objSession.OperatingSystem
```

See Also

Version Property (Session Object)

Outbox Property (Session Object)

Returns a Folder object representing the current user's default Outbox folder.

Syntax

objSession.**Outbox**

Data Type

Object

Comments

The property returns **Nothing** if the current user does not have or has not enabled the Outbox folder.

In addition to the formal collection and object hierarchy shown, OLE Messaging supports a simpler relationship between the Session object and certain common folders. This simplified structure more readily supports basic messaging-enabled applications.

Example

```
' from the function Session_Outbox
Dim objFolder As Object
' ...
    Set objFolder = objSession.Outbox
    If objFolder Is Nothing Then
        MsgBox "Failed to open Outbox"
        Exit Function
    End If
    MsgBox "Folder name = " & objFolder.Name
    Set objMessages = objFolder.Messages
```

See Also

Folder Object

Inbox Property (Session Object)

Version Property (Session Object)

Returns the version number of the OLE Messaging Library as a string, for example, "1.00". Read-only.

Syntax

objSession.**Version**

Data Type

String

Comments

The version number for the OLE Messaging Library is represented by a string in the form "*n.xx*," where *n* represents a major version number and *xx* represents a minor version number.

Example

```
' see the function Session_Version
    Dim objSession As Object
    Set objSession = CreateObject("MAPI.Session")
    ' error handling here...
    MsgBox "Version number is " & objSession.Version
MsgBox "Welcome to OLE Messaging version " & objSession.Version
```

See Also

OperatingSystem Property (Session Object)

References

The following published references provide additional information about Visual Basic, Visual Basic for Applications, and OLE.

- *Microsoft Visual Basic Programmer's Guide*, Chapter 23, "Programming Other Applications' Objects"
- *Excel Visual Basic for Applications Step by Step*, Microsoft Press
- *Microsoft Excel Visual Basic User's Guide*, Chapter 5, "Working with Objects in Visual Basic" and Chapter 10, "Controlling and Communicating with Other Applications"
- *Microsoft OLE 2 Programmer's Reference*, Microsoft Press
- *Inside OLE 2*, Microsoft Press

Note that this document contains the latest known information about the Microsoft OLE Messaging Library version 1.0 at the time of publication. Where terms in this document differ from other Visual Basic, OLE, or COM terms, this document should be viewed as the definition of the specific implementation represented by OLE Messaging Library version 1.0.

How Programmable Objects Work

How do programmable objects work? How does the OLE Messaging Library offer its powerful ability to create and manage messaging objects?

This section provides a *very* short introduction to the Microsoft Component Object Model, OLE Automation, and the OLE programmability interface **IDispatch**. For complete details, see the *OLE 2 Programmer's Reference*.

You do not need to understand this section in order to use the OLE Messaging Library.

COM Interfaces

With the combination of Microsoft RPC (Remote Procedure Call) and Microsoft OLE technology, Microsoft began to shift the C/C++ programming model from individual API functions, such as those offered in the Windows 3.1 SDK and Win32 SDK, to a distributed object model that is based on *interfaces*. An interface is simply a group of logically-related functions. Note that the interface consists only of functions. There are no facilities for directly accessing data within an interface, except through the functions.

The benefit of such a distributed object model is that it allows developers to create small, independent, self-managing software objects. This modular approach allows software functionality to be developed in small "building blocks" that are then fitted together. Your application no longer has to handle every possible data format or possible application feature, as long as it can be integrated with other objects that can handle the desired formats and features.

The notion of objects is very familiar to Visual Basic developers. Many software industry analysts have noted that the most visible success of object-oriented programming to date is the widespread use of Microsoft Visual Basic custom controls.

One of the benefits of the modular, interface-based approach to software development is that individual interfaces usually contain significantly fewer functions than libraries, with the promise of more efficient use of memory. Whenever you want to use one function in a library, the entire library must be loaded into memory. Splitting function libraries into smaller interfaces makes it more likely that you load only the functions that you actually need. (Or at least that you load fewer that you don't need.)

By convention, interface names start with the letter 'I'. The functions are given a specific ordering within the interface. Knowing the order of the functions is important for developers who must define their own *vtables*, or function dispatch tables. The C++ compiler creates vtables for you, but if you are writing in C, you must create your own.

The functions of an interface still physically reside in an .EXE or .DLL file, but Microsoft has defined new rules for how these files are registered on the system and how they are loaded and unloaded from memory. Microsoft refers to the new rules as the *Component Object Model,* or *COM*.

According to the rules, the first three functions in all interfaces are always **QueryInterface** (which developers call "QI"), **AddRef**, and **Release**. These functions provide a pointer to the interface when someone asks for it, keep track of the number of programs that are being served by the interface, and control how the physical .DLL or .EXE gets loaded and unloaded. Any other functions in the interface are defined by the person who creates the interface. The interface that consists of these three common functions, **QueryInterface, AddRef, and Release**, is called **IUnknown**. Developers can always obtain a pointer to an **IUnknown** object.

The component object model, like RPC before it, makes a strong distinction between the definition of the interface and its implementation. The interface functions and the data items that make up the parameters are defined in a very precise way, using a special language designed specifically for defining interfaces. These languages (such as MIDL, the Microsoft Interface Definition Language, and ODL, the Object Definition Language) do not allow you to use indefinite type names, such as **void ***, or types that change from computer to computer, such as **int**. The goal is to force you to specify the exact size of all data. This makes it possible for, say, one person to define an interface, a second person to implement the interface, and a third person to write a program that calls that interface.

Developers who write C and C++ code that use these types of interfaces read the object's interface definition language (IDL) files. They know exactly what functions are present in the interface and what data is required. They can call the interfaces directly.

For developers who are not writing in C and C++, or do not have access to the object's interface definition language files, Microsoft's component object model defines another way to use software components. This is based on an interface named **IDispatch**.

IDispatch

IDispatch is a COM interface that is designed in such a way that it can call virtually any other COM interface. Developers working in Visual Basic often cannot call COM interfaces directly, as they would from C or C++. However, when their tool supports **IDispatch**, as Visual Basic does, and when the object they want to call supports **IDispatch**, they can call its COM interfaces *indirectly*.

The main method offered by **IDispatch** is called **Invoke.** This method can be thought to add a level of indirection to the control flow of the Component Object Model. In the standard model, an object obtains a pointer to an interface and then calls a member function of the interface. **IDispatch** adds a level of indirection: Instead of directly calling the member function of the interface, the program calls **IDispatch::Invoke**, and **IDispatch::Invoke** calls the member function for you.

Invoke is a general method-calling machine. Its parameters include a value that identifies the method that is to be called and the parameters that are to be sent to it. In order to be able to handle the wide variety of parameters that other COM methods use as parameters, **Invoke** uses a self-describing data structure called a VARIANTARG.

The VARIANTARG structure contains two parts: a type field, which represents the data type, and the data field, which represents the actual value of the data. The values such as VT_I2, VT_I4, and so on, are the constants that define valid values for the data types.

Associated with **IDispatch** is the notion of a *type library*. The type library publishes information about an interface so that it is available to Visual Basic programs. The type library, or *typelib*, contains the same kind of information that C/C++ programmers would obtain from a header file: the name of the method and the sequence and types of its parameters.

An executable or DLL that exposes **IDispatch** and its type library is known as an *OLE Automation server*. The OLE Messaging Library is such a server.

OLE Messaging: An OLE Automation Server

So, let's put it all together, from the bottom up, to see how OLE Messaging works.

- Service providers implement COM interfaces—specifically, the Extended MAPI interfaces—as described in the Microsoft *MAPI Programmer's Reference*.
- The OLE Messaging Library implements several objects (Session, Message, etc.) that act as *clients* to these Extended MAPI interfaces. That is, the OLE Messaging objects obtain pointers to the Extended MAPI interfaces and call methods.
- The OLE Messaging Library implements **IDispatch** and acts as an OLE Automation *server* so that it can be called by tools that can use **IDispatch**, such as Visual Basic. That is, the OLE Messaging Library allows other programs to call its **IDispatch** interface. It provides its own registration (.REG) file so that it can be registered on a computer as an OLE Automation server.
- The OLE Messaging Library publishes a type library that contains information about the objects that it makes available through **IDispatch**.
- Your Visual Basic application acts as a client to the OLE Messaging Library. It reads the OLE Messaging Library's type library to obtain information about the objects, methods, and properties. When your VB application declares a variable as an object (with code such as "Dim objSession as Object") and uses that object's properties and methods (with code such as "MsgBox objSession.Class"), Visual Basic makes calls to **IDispatch** on your behalf.

The relationships between these programs are shown in the following diagram:

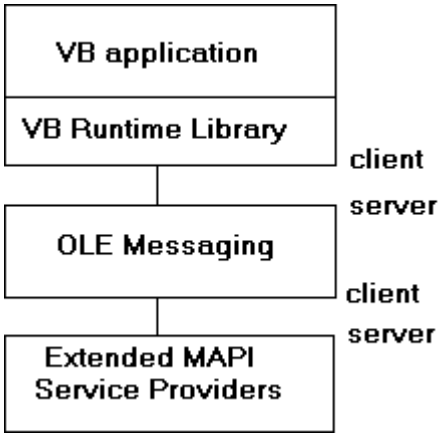


Figure 1. Visual Basic is a client to the OLE Automation server, the OLE Messaging Library. The OLE Messaging Library, in turn, acts as a client to the Extended MAPI services.

The OLE Messaging Library and Extended MAPI

The OLE Messaging Library calls Microsoft Extended MAPI interfaces for you. The following table describes the Extended MAPI interfaces that OLE Messaging calls when you manipulate an OLE Messaging object:

OLE Messaging object	Extended MAPI or OLE interface called by OLE Messaging
AddressEntry	IABContainer, IMAPIProp
Attachment	IAttach
Field	IStream, IMAPIProp
Folder	IMAPIFolder
Message	IMessage
Recipient	IMAPIProp
Session	IMAPISession

For collection objects, the OLE Messaging Library calls the Extended MAPI interface **IMAPITable.**

The OLE Messaging Library also calls the Extended MAPI interface **IMAPIProp**. Many of the properties exposed by the OLE Messaging Library are based on Extended MAPI properties. The following table describes the mapping between some OLE Messaging Library properties and the underlying MAPI properties:

OLE Messaging Library object	Property	Extended MAPI property	Extended MAPI property type
AddressEntry	Address	PR_EMAIL_ADDRESS	PT_TSTRING
AddressEntry	ID	PR_ENTRYID	PT_BINARY
AddressEntry	Name	PR_DISPLAY_NAME	PT_TSTRING
AddressEntry	Type	PR_ADDRTYPE	PT_TSTRING
Attachment	Index	PR_ATTACH_NUM	PT_LONG
Attachment	Name	PR_ATTACH_FILENAME	PT_TSTRING
Attachment	Position	PR_RENDERING_POSITION	PT_LONG
Attachment	Source	PR_ATTACH_PATHNAME	PT_TSTRING
Attachment	Type	PR_ATTACH_METHOD	PT_LONG
Folder	FolderID	PR_PARENT_ENTRYID	PT_BINARY
Folder	ID	PR_ENTRYID	PT_BINARY
Folder	Name	PR_DISPLAY_NAME	PT_TSTRING
Folder	StoreID	PR_STORE_ENTRYID	PT_BINARY
Message	Conversation	PR_CONVERSATION_	PT_BINARY

		KEY	
Message	ConversationIndex	PR_CONVERSATION_INDEX	PT_BINARY
Message	ConversationTopic	PR_CONVERSATION_TOPIC	PT_STRING
Message	DeliveryReceipt	PR_ORIGINATOR_DELIVERY_REPORT_REQUESTED	PT_BOOLEAN
Message	Encrypted	PR_SECURITY	PT_LONG
Message	FolderID	PR_PARENT_ENTRYID	PT_BINARY
Message	ID	PR_ENTRYID	PT_BINARY
Message	Importance	PR_IMPORTANCE	PT_LONG
Message	ReadReceipt	PR_READ_RECEIPT_REQUESTED	PT_BOOLEAN
Message	Sender	PR_SENDER_ENTRYID	PT_BINARY
Message	Sent	PR_MESSAGE_FLAGS	PT_LONG
Message	Signed	PR_SECURITY	PT_LONG
Message	Size	PR_MESSAGE_SIZE	PT_LONG
Message	StoreID	PR_STORE_ENTRYID	PT_BINARY
Message	Subject	PR_SUBJECT	PT_TSTRING
Message	Submitted	PR_MESSAGE_FLAGS	PT_LONG
Message	Text	PR_BODY	PT_TSTRING
Message	TimeReceived	PR_MESSAGE_DELIVERY_TIME	PT_SYSTIME
Message	TimeSent	PR_CLIENT_SUBMIT_TIME	PT_SYSTIME
Message	Type	PR_MESSAGE_CLASS	PT_TSTRING
Message	Unread	PR_MESSAGE_FLAGS	PT_LONG
Recipient	Name	PR_DISPLAY_NAME	PT_TSTRING
Recipient	Type	PR_RECIPIENT_TYPE	PT_LONG
Session	Name	PR_DISPLAY_NAME	PT_TSTRING

For more information about Extended MAPI properties, see the Microsoft *MAPI Programmer's Reference*.