Microsoft OLE Messaging Library Programmer's Reference

Introduction

Overview of OLE Messaging

Programmer's Guide

Supplemental Information

- **Error Codes**
- **How Programmable Objects Work**
- COM Interfaces
- IDispatch
- The OLE Messaging Library: An Automation Server
- The OLE Messaging Library and MAPI
- Additional References

Objects

Methods Properties Summary

- Objects, Properties, and Methods
- Object Model
- Properties Common to All OLE Messaging Library Objects
- AddressEntries Collection Object
- AddressEntry Object
- AddressEntryFilter Object
- AddressList Object
- AddressLists Collection Object
- Attachment Object
- Attachments Collection Object
- Field Object
- Fields Collection Object
- Folder Object
- Folders Collection Object
- InfoStore Object
- InfoStores Collection Object
- Message Object
- MessageFilter Object
- Messages Collection Object
- Recipient Object
- Recipients Collection Object
- Session Object

Methods

- IsSameAs Method (AddressList Object)
- IsSameAs Method (Attachment Object)
- IsSameAs Method (Folder Object)
- IsSameAs Method (InfoStore Object)
- IsSameAs Method (Message Object)
- IsSameAs Method (MessageFilter Object)
- IsSameAs Method (Recipient Object)
- Logoff Method (Session Object)
- Logon Method (Session Object)
- MoveTo Method (Folder Object)
- MoveTo Method (Message Object)
- Options Method (Message Object)
- ReadFromFile Method (Attachment Object)
- ReadFromFile Method (Field Object)
- Resolve Method (Recipient Object)
- Resolve Method (Recipients Collection)
- Send Method (Message Object)
- SetNamespace Method (Fields Collection)
- Sort Method (AddressEntries Collection)
- Sort Method (Folders Collection)
- Sort Method (Messages Collection)
- Update Method (AddressEntry Object)
- Update Method (Folder Object)
- Update Method (Message Object)
- WriteToFile Method (Attachment Object)
- WriteToFile Method (Field Object)

Properties

Methods Objects Summary

- Address Property (AddressEntry Object)
- Address Property (AddressEntryFilter Object)
- Address Property (Recipient Object)
- AddressEntries Property (AddressList Object)
- AddressEntry Property (Recipient Object)
- AddressLists Property (Session Object)
- Application Property (All OLE Messaging Library Objects)
- Attachments Property (Message Object)
- Class Property (All OLE Messaging Library Objects)
- Conversation Property (Message Object)
- Conversation Property (MessageFilter Object)
- ConversationIndex Property (Message Object)
- ConversationTopic Property (Message Object)
- Count Property (AddressEntries Collection)
- Count Property (AddressLists Collection)
- Count Property (Attachments Collection)
- Count Property (Fields Collection)
- Count Property (Folders Collection)
- Count Property (InfoStores Collection)
- Count Property (Messages Collection)
- Count Property (Recipients Collection)
- CurrentUser Property (Session Object)
- DeliveryReceipt Property (Message Object)
- DisplayType Property (AddressEntry Object)
- DisplayType Property (Recipient Object)
- Encrypted Property (Message Object)
- Fields Property (AddressEntry Object)
- Fields Property (AddressEntryFilter Object)
- Fields Property (Attachment Object)
- Fields Property (Folder Object)
- Fields Property (Message Object)
- Fields Property (MessageFilter Object)
- Filter Property (AddressEntries Collection)
- Filter Property (Messages Collection)
- FolderID Property (Folder Object)
- FolderID Property (Message Object)
- Folders Property (Folder Object)
- ID Property (AddressEntry Object)
- ID Property (AddressList Object)
- ID Property (Field Object)
- ID Property (Folder Object)
- ID Property (InfoStore Object)

Programmer's Reference Summary

- A -

- B -

- C -

- D -

- T -

- U -

- V -

- W -

- X -

- Y -

- Z -

Legal Information

Microsoft OLE Messaging Library Programmer's Reference

Introduction

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

The OLE 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 quickly build custom applications that cover all your business needs. For example, with these powerful building blocks you can build a custom application that allows your users to extract information from a database, copy it to a spreadsheet for analysis, then create a report with the results and mail the report to several people.

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

This guide assumes that you are familiar with the Microsoft Visual Basic programming model. To help you use the OLE Messaging Library, this guide 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 and a tool that supports Automation. 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 version 3.0 does not support multivalued properties.

Quick Start

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

In this code fragment, we first create a Session object and log on. We then create a Message object and set its properties to indicate its subject and content. Next we create a Recipient object and call its **Resolve** method to obtain a full messaging address. We then call the Message object's **Send** method to transmit the message. Finally, we display a completion message and log off.

```
' You must install the MAPI SDK, registering the
' OLE Messaging Library, to run this sample code.
' This sample uses Visual Basic 3.0 error handling.
'
Function QuickStart()
Dim objSession As Session    ' Session object
Dim objMessage As Message    ' Message object
Dim objOneRecip As Recipient ' Recipient object

On Error GoTo error_olemsg

' create a session and log on -- username and password in profile
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 ' get MAPI to determine complete e-mail address

' send the message and log off
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. This code fragmnent 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 Installation

The OLE Messaging Library is installed with the MAPI Software Development Kit (SDK). The MAPI SDK setup program registers the OLE Messaging Library for subsequent use by tools that support Automation.

Note In the current release, the OLE Messaging Library is installed only as part of the MAPI SDK. No separate setup program is provided.

When you use the OLE Messaging Library with a tool that supports Automation, verify that the tool has referenced the OLE Messaging Library. For example, when you are using Microsoft Visual Basic version 4.0, choose the **References** command from the **Tools** menu, and select the check box for **OLE/Messaging 1.0 Object Library**.

When the OLE Messaging Library is available, the following flag is set in the WIN.INI file:

```
[Mail]
OLEMessaging=1
```

The **OLEMsgPersistenceTimeout** registry setting controls how quickly the OLE Messaging Library shuts down and unloads from memory after all messaging objects are released by client applications. On Win32® systems, the setting appears at the following registry location:

HKEY_LOCAL_MACHINE\Software\Microsoft\Windows Messaging Subsystem

For 16-bit Microsoft® Windows® systems, the **OLEMsgPersistenceTimeout** setting appears within the [MAPI] section of the WIN.INI file.

About This Guide

Overview defines the MAPI terms used in this guide 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.

Programming Tasks offers sample Visual Basic code for many common programming tasks, such as creating and sending a message, posting a message to a public folder, navigating through folders, searching through address books, and handling errors.

Objects, Properties, and Methods contains comprehensive reference information for the properties and methods of all objects and collection objects.

The appendixes, Error Codes and How Programmable Objects Work, offer additional background information about Automation, the technology used by the OLE Messaging Library.

The best way to learn about the OLE Messaging Library is to alternate your reading with hands-on programming. You can use the sample code that is provided with the OLE Messaging Library.

Overview

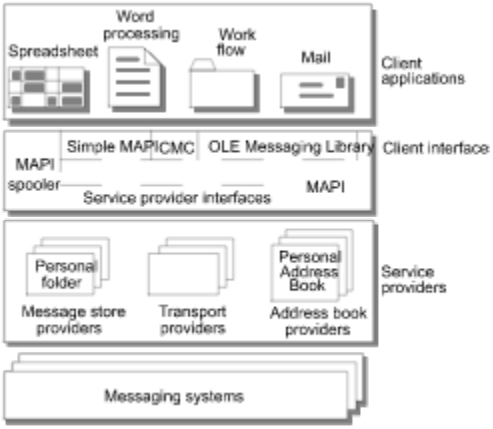
This section offers a brief introduction to MAPI and describes how the OLE Messaging Library fits into the mix of MAPI programming interfaces. It provides a short description of Automation, which is the basis of the design of the OLE Messaging Library. The section concludes with a conceptual overview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

Introduction to MAPI

MAPI 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.

The MAPI architecture can 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.

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



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, transports, and message stores.

An *address book* service is similar to a telephone directory or Yellow Pages. The address book can be thought of as a persistent 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.

A *transport* supports communication between different devices and different underlying messaging systems.

A *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, that 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 one or more attachments, which are attached to and sent with the message. An *attachment* can be the contents of a file, a link to a file, an OLE object, or another message embedded in this message.

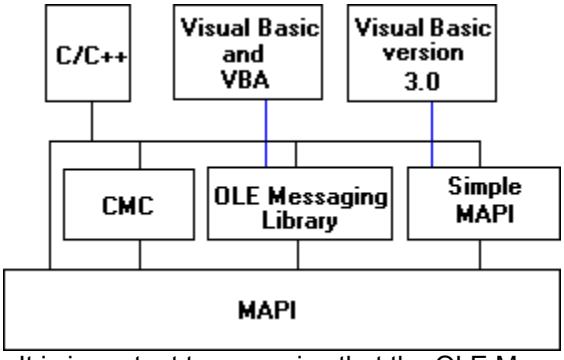
Several *properties* can be associated with a message: its subject, its importance, its delivery properties (such as the time it is sent and received), and whether to notify the sender when the message is delivered and read. Some message properties identify the message as part of a *conversation*. The 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.* A 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 should *resolve* each recipient; this means you should check each recipient against the address book to make sure its e-mail address is valid.

MAPI Programming Interfaces

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

The following figure shows the OLE Messaging Library as a layer that is built on top of MAPI. This is similar to the way function calls to the Common Messaging Calls (CMC) interface are mapped to the underlying MAPI interfaces. It also demonstrates that the OLE Messaging Library is the only interface available to all the concerned languages, namely Visual Basic, Visual Basic for Applications (VBA), and C/C++.



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

The following table summarizes the programming interfaces that Microsoft provides for MAPI.

Programming interface	**Description**
MAPI custom controls	User interface elements for Visual Basic version 3.0 developers.
Simple MAPI	Functions for Visual Basic version 3.0 and C/C++ client developers that allow access to the Inbox (no access to MAPI properties). Most developers should probably use either CMC or MAPI rather than Simple MAPI.
OLE Messaging Library	Programmable messaging objects for Visual Basic/VBA and C/C++ developers.
Common Messaging Calls (CMC)	Functions for C/C++ client developers; X.400 API Association (XAPIA) standard.
MAPI	OLE Component Object Model (COM) interfaces for C/C++ developers. Full access to all MAPI programming interfaces. Implemented and called by clients, service providers, and MAPI itself.

MAPI Custom Controls and the OLE Messaging Library

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 convenient for offering more specialized capabilities than are provided by 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 and queries. As the data access library lets you use database objects, the OLE Messaging Library lets you add messaging to your applications.

MAPI Functions and the OLE Messaging Library

Compared to the function call interfaces of traditional application programming interface (API) libraries, an Automation object library yields faster development and code that is easier to read, debug, and maintain.

The OLE Messaging 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 Library interface.

Task or code	**Function call interface**	**OLE Messaging Library**
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 redimensioning 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 its own set of error codes.	Integrated with Visual Basic error handling during both design and run time.
Return values	Returned implicitly in the parameters of the function call.	Returned as an explicit result of a method or in 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.

Introduction to Automation

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

For the purposes of this document, an *object* is an 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 programs. 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 “LoadDatabase”. They are general-purpose functions that enable an open-ended number of programmable objects, including the OLE Messaging Library.

Throughout this section, Visual Basic is used as a concrete example of a tool that supports 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 Automation, such as the libraries of programmable objects provided by Microsoft Excel version 5.0 and Microsoft Access version 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 containing the results of those calculations and send the message to several recipients.

OLE Messaging Library Object Design

The OLE Messaging Library is designed for ease of use and convenience. It implements the 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 Introduction to MAPI.)

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

The OLE Messaging Library defines the following objects:

- AddressEntries collection
- AddressEntry
- AddressEntryFilter
- AddressList
- AddressLists collection
- Attachment
- Attachments collection
- Field
- Fields collection
- Folder
- Folders collection
- InfoStore
- InfoStores collection
- Message
- MessageFilter
- Messages collection
- Recipient
- Recipients collection
- Session

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

- **High-level objects**, which can be created directly in a Visual Basic program.
- **Child objects**, which are created automatically when the high-level objects are created.
- **Collections**, or groups of objects of the same type.

High-Level Objects

The high-level, or top-level, objects are the Session, Folder, and Message objects. Other objects are accessible only through these high-level objects.

You can create a Session object either through early binding:

```
Dim objSession as MAPI.Session
objSession.Logon
```

or through late binding:

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

and then use the **Logon** method to initiate a session with MAPI.

Note that early binding is not supported in OLE Messaging Library versions previous to 1.1.

C/C++ programmers use globally unique identifiers (GUIDs) for these objects, defined in the type library for the OLE Messaging Library. The following C++ code fragment demonstrates how to create a Session object and call its **Logon** method:

```
// create a Session object and log on using IDispatch interface
// to the OLE Messaging library
#include <ole2.h>
#include <stdio.h>
#include <stdlib.h>  // for exit
#define dispidM_Logon 119  // get constants for all props, methods
// allows you to save cost of GetIdsFromNames calls
// can generate yourself by calling GetIdsFromNames for all
// properties and methods
// GUID values for Session defined in the type library
static const CLSID GUID_OM_SESSION =
{0x3FA7DEB3, 0x6438, 0x101B, {0xAC, 0xC1, 0, 0xAA, 0, 0x42, 0x33, 0x26}};
void main(void)
{
HRESULT hr;

/* interface pointers */
LPUNKNOWN punk = NULL; // IUnknown *; used to get IDispatch *
DISPPARAMS dispparamsNoArgs = {NULL, NULL, 0, 0};
VARIANT varRetVal;
IDispatch * pSession;

    //Initialize OLE.
    hr = OleInitialize(NULL);
     printf("OleInitialize returned 0x%lx\n", hr);
    VariantInit(&varRetVal);
// Create an instance of the OLE Messaging Library Session object
// Ask for its IDispatch interface.
    hr = CoCreateInstance(GUID_OM_SESSION,
                          NULL,
                          CLSCTX_SERVER,
                          IID_IUnknown,
                          (void FAR* FAR*)&punk);
```

```
    printf("CoCreateInstance returned 0x%lx\n", hr);
    if (S_OK != hr)
        exit(1);
    hr = punk->QueryInterface(IID_IDispatch, (void FAR* FAR*)&pSession);
    punk->Release();       // no longer needed; release it
    printf("QI for IID_IDispatch returned 0x%lx\n", hr);
    if (S_OK != hr)
        exit(1);
// Logon using the session object; call its Logon method
    hr = pSession->Invoke(dispidM_Logon, // value = 119
                          IID_NULL,
                          LOCALE_SYSTEM_DEFAULT,
                          DISPATCH_METHOD,
                          &dispparamsNoArgs,
                          &varRetVal,
                          NULL,
                          NULL);
    printf("Invoke returned 0x%lx\n", hr);
    printf("Logon call returned 0x%lx\n", varRetVal.lVal);
// do other things here...
// when done, release the Session dispatch object and shut down OLE
    pSession->Release();
    OleUninitialize();
```

The following table lists the GUIDs for the high-level objects accessible to C/C++ programmers. Note the close relationship; only the fourth of the 16 bytes differs among the GUIDs.

OLE Messaging Library object	**GUID**
Session object	3FA7DEB36438101BACC100AA00423326
Folder object	3FA7DEB56438101BACC100AA00423326
Message object	3FA7DEB46438101BACC100AA00423326

Child Objects

All OLE Messaging Library objects can be considered as relative to a Session object. A session's immediate child objects are the AddressLists collection object, the InfoStores collection object, and the Inbox or Outbox Folder object. These provide access, respectively, to the root of the address book hierarchy for the current session, the set of all message stores available to the session, and the current default Inbox and Outbox folders.

The session's child objects have their own child objects, which in turn have child objects, and so on. This hierarchy permits increasingly detailed levels of access. The AddressLists collection, for example, contains one or more AddressList child objects, each representing one available address book container. Each of these has as its child an AddressEntries collection containing AddressEntry child objects. Each address entry that is a distribution list has a **Members** property that provides another AddressEntries collection for the members of the distribution list.

See the Object Model diagram for the logical hierarchy of the OLE Messaging Library.

In addition to the hierarchy of objects, each object has properties and methods. 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. For example,

```
objSession.AddressLists("Personal Address Book").AddressEntriesColl(2)
```

refers to the second AddressEntry object in the AddressEntries collection of the current session's personal address book (PAB) AddressList object.

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 Library object. For example, the Messages collection is the name of the collection that contains Message objects. The OLE Messaging Library supports the following collections:

- AddressEntries
- AddressLists
- Attachments
- Fields
- Folders
- InfoStores
- Messages
- Recipients

For purposes of accessing their individual member objects, collections can be characterized as either large or small.

For a *small collection*, the service provider maintains an accurate count of the number of objects in the collection. The AddressLists, Attachments, Fields, InfoStores, and Recipients collections are considered small collections. You can access individual items using an index into the collection. You can also add and delete items from the collection (except for the AddressLists and InfoStores collections, which are read-only for the OLE Messaging Library).

Small collections, with a known number of member objects, can rely on their **Count** and **Item** properties. The **Count** property holds at all times the current number of member objects, and the **Item** property can select any arbitrary member of the collection. A small collection also has an implied temporary **Index** property, 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 **Index** value for the first member object is 1.

For example, in an Attachments collection with three Attachment objects, 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 can add and delete objects while your application is running. The **Count** property is not updated until you re-create or refresh the collection, for example by calling the parent Message object's **Update** or **Send** method. The attachment is saved in the MAPI system when you refresh the Message object, and the **Count** properties of its Attachments and Recipients collections are updated.

For a *large collection*, the service provider cannot always maintain an accurate count of member objects. The AddressEntries, Folders, and Messages collections are considered large collections. In preference to using a count, these collections support **Get** methods that let you get the first, last, next, or previous item in the collection. Programmers needing to access individual objects in a large collection are strongly advised to use the Visual Basic **For Each** statement or the **Get** methods.

Large collections, with an uncertain number of member objects, support the **Count** and **Item** properties in a limited way. The **Count** property can tell you if the collection is empty or not, but it cannot be used as the collection's exact size if it has a very large value such as **mapiMaxCount**. The **Item** property satisfies syntactical requirements in an indexed loop but cannot select an arbitrary member of the collection. For more information on using the **Count** and **Item** properties in a large collection, see the example in the **Count** property.

MAPI assigns a permanent, unique string **ID** property when an individual member object is created. These identifiers do not change from one MAPI session to another. You can call the Session object's **GetAddressEntry**, **GetFolder**, or **GetMessage** methods, specifying the unique identifier, to obtain the individual AddressEntry, 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 fragment 1 causes the OLE Messaging Library to create a new Messages collection and to reinitialize the value of the collection's "current message" in each **Set** statement. The **GetFirst** and **GetNext** method calls return the same value for *objMessage*.

Code fragment 2 uses the existing collection *objMsgColl*, so the **GetFirst** and **GetNext** calls function 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 you can perform with the OLE Messaging Library. The first task your application must do is obtain a valid Session object as described in Starting an OLE Messaging Session.

Category	Programming tasks
General programming tasks	Handling Errors Improving Application Performance Starting an OLE Messaging Session Viewing MAPI Properties
Working with messages	Adding Attachments to a Message Checking for New Mail Creating and Sending a Message Customizing a Folder or Message Deleting a Message Filtering Messages in a Folder 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 Creating a New Address Book 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

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 navigate 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. 3. Call the Message object's **Update** or **Send** method.
Changing an Existing	1. Obtain a valid AddressEntry object.

Address Entry	2. Update the **Name**, **Type**, or **Address** property. 3. Call the AddressEntry object's **Update** method.
Checking for New Mail	Count messages in the Inbox folder that have the **Unread** property set to **True**. - or - 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 source Message object's **CopyTo** method. 3. Call the new Message object's **Update** method.
Creating a New Address Book Entry	1. Obtain the Session object's AddressLists collection. 2. Select the AddressList object corresponding to the desired address book container. 3. Obtain the address list's AddressEntries collection. 4. Call the collection's **Add** 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. - or - 3. Copy a Recipients collection from another message to the new message's **Recipients** property. 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.
Filtering Messages in a Folder	1. Access the Folder in which you wish to filter the messages. 2. Obtain the MessageFilter object for the Folder.

	3. Select and set the desired MessageFilter properties to specify the filter.
Handling Errors	Use the Microsoft Visual Basic **On Error Goto** statement 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	1. Obtain the source message that you want to move. 2. Call the source Message object's **MoveTo** method. 3. Call the Message object's **Update** method at its new location.
Posting Messages to a Public Folder	Use a procedure similar to 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 identifier value. - or - Call the Folders collection's **Get** methods to get individual Folder objects, and compare properties of each folder with the desired

	property values.
Searching for a Message	Use the Session object's **GetMessage** method to obtain the message from its known identifier value. - or - Call the Messages collection's **Get** methods to get individual Message objects, using a message filter to reduce the number of messages searched, and if necessary compare properties of each message with the desired property values.
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 address book dialog box. 2. Set a Recipients collection object to the Recipients collection returned by the **AddressBook** method. 3. Use that Recipients collection or copy individual recipients from it.
Starting an OLE Messaging Session	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 a MAPI property tag as the Fields collection's **Item** property.
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**.

It is important to understand the hierarchy of the OLE Messaging Library objects, 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 guide, 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 convention is used in an attempt to make the material easy to read.

Accessing Folders

Folders can be organized in a hierarchy, allowing you to access folders within folders. Subfolders appear in the Folders collection returned by the **Folders** property of the Folder object containing them.

With the OLE Messaging Library version 1.1 and later, you can create a new folder within an existing folder using the **Add** method of the Folders collection.

There are two general approaches for accessing folders:

- Obtaining the folder directly by calling the Session object's **GetFolder** method.
- Navigating folders using the Folders collection's **Get** methods.

To obtain the folder directly using the **GetFolder** method, you must have the folder's identifier. In the following code fragment, 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 navigate 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 collection's **GetFirst** and **GetNext** methods to get each Folder object 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 code fragment navigates 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 object is a Folder
            ' with OLE Messaging Library 1.1, can use Class value:
            ' If mapiFolder = objFolder.Class Then
        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 Folders ' the child Folders collection
Dim objOneSubfolder As Folder ' 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 obtained from the Message object's **Attachments** property. The relationship between the Message object and an attachment is shown here:

Message object
  Attachments collection
    Attachment object
      Type property
      Source property

The OLE Messaging Library supports several different kinds of attachments: files, links to files, OLE objects, and embedded messages. An attachment's type is specified by its **Type** property. To add an attachment, use the related Attachment object property or method appropriate for that type, as shown in the following table:

Attachment type	Related Attachment object property or method
mapiFileData	**ReadFromFile** method
mapiFileLink	**Source** property
mapiOLE	**ReadFromFile** method
mapiEmbeddedMessage	**ID** property of the Message object to be embedded

The following example demonstrates inserting a file as an attachment. 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 Starting an OLE Messaging Session.

```
' 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 Message  ' local
Dim objRecip As Recipient  ' 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."
        .Text = " " & objMessage.Text ' add placeholder for attachment
        Set objAttach = .Attachments.Add ' add the attachment
        If objAttach Is Nothing Then
            MsgBox "Unable to create new Attachment object"
            Exit Function
```

```
        End If
        With objAttach
            .Type = mapiFileData
            .Position = 0 ' render at first character of message
            .Name = "c:\smiley.bmp"
            .ReadFromFile "c:\smiley.bmp"
         End With
         objAttach.Name = "smiley.bmp"
         .Update ' update message to save attachment in MAPI system
    End With
    MsgBox "Created message, added 1 mapiFileData attachment, updated"
    Exit Function

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

End Function
```

The attachment overwrites the placeholder character at the rendering position specified by the attachment's **Position** property. A space is normally used for the placeholder character.

The OLE Messaging Library does not actually place the attachment within the message; that is the responsibility of the messaging client application. You can also use the value -1 for the **Position** property, which indicates that the attachment should be sent with the message, but should not be rendered via the **Position** property.

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 version 7.0 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 the attachment
    With objAttach
        .Type = mapiFileLink
        .Position = 0 ' render at first character of message
        .Source = "\\server\bitmaps\honey.bmp"
    End With
    ' must update the message to save the new info
    objOneMsg.Update ' update message; save attachment in MAPI system
    MsgBox "Added an attachment of type mapiFileLink"
    Exit Function
```

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

End Function
```

See Also

Creating and Sending a Message

Changing an Existing Address Entry

The OLE Messaging Library lets you change existing address entries in any address book container for which you have modification permission. Typically you have such permission only for your personal address book (PAB).

To change an existing address entry

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.
   - Use the Message object's **Recipients** property to obtain a Recipients collection. Then obtain an individual Recipient object and use its **AddressEntry** property to obtain its child AddressEntry object.
2. Change individual properties of the AddressEntry object, such as the **Address**, **Name**, or **Type** property.
3. Call the AddressEntry object's **Update** method.

The following sample code demonstrates this procedure:

```
' Function: AddressEntry_Update
' Purpose: Demonstrate the Update method
' See documentation topic: Update method AddressEntry object
Function AddressEntry_Update()
Dim objRecipColl As Recipients  ' Recipients collection
Dim objNewRecip As Recipient    ' New recipient object

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 address entry name: " & objNewRecip.AddressEntry.Name
Exit Function

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

End Function
```

See Also

Using Addresses, Creating a New Address Book Entry, Selecting Recipients from the Address Book

Checking for New Mail

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

You can force immediate delivery of any pending messages by calling the Session object's **DeliverNow** method.

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

```
' Function: Util_CountUnread
' Purpose:  Count unread messages in a folder
'
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 increments its count of new messages and updates the saved value.

With the OLE Messaging Library version 1.1 or later, you can use the Messages collection's **Filter** property to obtain a MessageFilter object. Setting the message filter's **TimeFirst** or **Unread** property reduces the number of messages presented to the loop doing the counting or other processing of new messages.

See Also

Filtering Messages in a Folder, Reading a Message from the Inbox

Copying a Message to Another Folder

The procedure documented in this section demonstrates, first, the old way to copy message properties using the Messages collection's **Add** method, and then how to take advantage of the newer **CopyTo** method of the Message object.

Note With OLE Messaging Library version 1.0, the Message object's **Sender** property and other read-only properties of the Message object were not preserved during the first part of the procedure in this section. To preserve these properties using the old procedure, you had to append their text fields to read/write properties, such as the Message object's **Text** property.

With the **CopyTo** method, every property that is set on a Message object is automatically copied to the new Message object, regardless of whether it has read-only or read/write access. The access of every property is also preserved across the copy.

To copy a message from one folder to another folder using the OLE Messaging Library

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.

   - or -

   Call the source Message object's **CopyTo** method.
3. Call the new Message object's **Update** method to save all new information in the MAPI system.

The hierarchy of objects is as follows:

Session object
    Folder object (Inbox or Outbox)
        Messages collection
            Message object
    InfoStores collection
        InfoStore 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's Messages collection. For more information about finding messages, see 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 Searching for a Folder.

The following example copies the first two messages in the given folder to the Inbox. They could as easily be copied to any folder with a known identifier and therefore accessible using the Session object's **GetFolder** method. The example uses the old procedure to copy the first message and the new **CopyTo** method to copy the second.

This code fragment assumes that the application has already created the Session object variable *objSession* and successfully called the Session object's **Logon** method, as described in Starting an

OLE Messaging Session.

```
'/*******************************/
' 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 messages to copy
' for this sample, just copy the first two messages to the Inbox
' assume session object already created, validated, and logged on
Dim objMsgColl As Messages   ' given folder's Messages collection
Dim objThisMsg As Message    ' original message from given folder
Dim objInbox As Folder       ' destination folder is Inbox
Dim objCopyMsg As Message    ' new message that is the copy
Dim objOneRecip As Recipient ' single message recipient being copied
Dim strRecipName As String   ' recipient name from original message
Dim i As Integer             ' loop counter

On Error GoTo error_olemsg
If objGivenFolder Is Nothing Then
    MsgBox "Must supply a valid folder"
    Exit Function
End If
Set objMsgColl = objGivenFolder.Messages ' will be reused later
' ( ... then validate the Messages collection before proceeding ... )
Set objThisMsg = objMsgColl.GetFirst() ' filter parameter not needed
If objThisMsg Is Nothing Then
    MsgBox "No valid messages in given folder"
    Exit Function
End If
' Get Inbox as destination folder
Set objInbox = objSession.Inbox
If objInbox Is Nothing Then
    MsgBox "Unable to open Inbox"
    Exit Function
Else
    MsgBox "Copying first message to Inbox"
End If
' Copy first message using old procedure
Set objCopyMsg = objInbox.Messages.Add _
   (Subject:=objThisMsg.Subject, _
    Text:=objThisMsg.Text, _
    Type:=objThisMsg.Type, _
    Importance:=objThisMsg.Importance)
If objCopyMsg Is Nothing Then
    MsgBox "Unable to create new message in Inbox"
    Exit Function
End If
' Copy all the recipients
For i = 1 To objThisMsg.Recipients.Count Step 1
    strRecipName = objThisMsg.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 = objThisMsg.Sent
objCopyMsg.Text = objThisMsg.Text
objCopyMsg.Unread = objThisMsg.Unread
' Update new message so all changes are saved in MAPI system
objCopyMsg.Update
' If MOVING a message to another folder, delete the original message:
'     objThisMsg.Delete
' Move operation implies that the original message is removed

' Now copy second message using new procedure
Set objThisMsg = objMsgColl.GetNext ()
' ( ... then validate the second message before proceeding ... )
Set objCopyMsg = objThisMsg.CopyTo (objInbox.ID)
' Then Update and we're done
objCopyMsg.Update
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 the old procedure does not preserve all message properties. The **CopyTo** method copies all properties with their values and access capabilities (read-only or read/write) unchanged.

See Also

Moving a Message to Another Folder

Creating a New Address Book Entry

You can create new address entries in a collection with the OLE Messaging Library version 1.1 and later.

You need permission to **Add** a new entry to an address book container. Usually you only have this permission for your personal address book (PAB).

For address entries in an address book container, the hierarchy of objects is as follows:

Session object
    AddressLists collection
        AddressList object
            AddressEntries collection
                AddressEntry object
                    Fields collection
                        Field object

The procedure is basically to work down the hierarchy. Once a session is established and logged on, you use the Session object's **AddressLists** property to obtain the AddressLists collection, select the AddressList object corresponding to the desired address book container, and use the address list's **AddressEntries** property to call the AddressEntries collection's **Add** method.

If you have not specified all the parameters in the call to the **Add** method, you can then supply the missing values by setting AddressEntry object properties such as **Address**, **Name**, and **Type**. You can also set MAPI properties and custom properties using the new address entry's **Fields** property. To create a custom property you call the Fields collection's **Add** method.

Finally, you commit all the new data to the address book container and to the MAPI system by calling the new address entry's **Update** method.

This code fragment adds a new entry to a user's personal address book (PAB). Note the use of early binding and of default properties. The objects are declared using early binding to reduce the amount of additional code necessary. The **Item** property is the default property of all collections and so does not need to be specifically referenced in the statements selecting items from the AddressLists and Fields collections.

```
' we assume we have add permission for our PAB
Function AddEntry()

Dim objSession As MAPI.Session   ' Session object
Dim objMyPAB As AddressList      ' personal address book object
Dim objNewEntry As AddressEntry  ' new address entry object

On Error GoTo error_olemsg

' log on to session, supplying username and password
objSession.Logon 'profileName:="MyProfile", _
                 'profilePassword:="my_password"

' get PAB AddressList from AddressLists collection of Session
Set objMyPAB = MAPI.Session.AddressLists("Personal Address Book")
If objMyPAB Is Nothing Then
    MsgBox "Invalid PAB from session"
    Exit Function
End If
```

```
' add new AddressEntry to AddressEntries collection of AddressList
Set objNewEntry = objMyPAB.AddressEntries.Add “SMTP”, “Jane Doe”
objNewEntry.Address = “janed@exchange.microsoft.com”

' set MAPI property in new AddressEntry
' (&H3A08001E is MAPI property tag for PR_BUSINESS_TELEPHONE_NUMBER)
objNewEntry.Fields(&H3A08001E) = “+1-206-555-9901”

' add custom property to new AddressEntry and set its value
objNewEntry.Fields.Add “CellularPhone”, vbString
objNewEntry.Fields(“CellularPhone”) = “+1-206-555-9902”

' commit new entry, properties, fields, and values to PAB AddressList
objNewEntry.Update
MsgBox “New address book entry successfully added”
Exit Function

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

End Function
```

Creating and Sending a Message

Creating and sending a message is easy when you use the OLE Messaging Library.

To create and send a message

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**, **Text**, and other properties.
4. Call the Recipients collection's **Add** method for each recipient, or copy the **Recipients** property from an existing message to the new message.
5. If necessary, set each Recipient object's **Address**, **AddressEntry**, and **Name** properties.
6. Call each Recipient object's **Resolve** method to validate the address information.
7. Call the Message object's **Send** method.

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

```
' This also appears as the "QuickStart" example in "Overview"
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 an object other than the Message object, save your changes using the **Update** method before you clear or reuse the variable that refers to the object. 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 supports the use of fields with the AddressEntry, AddressEntryFilter, Attachment, Folder, Message and MessageFilter objects. These objects all have a **Fields** property through which the Fields collection can be accessed.

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 code fragment shows how to add the “Keyword” field to a 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 Field    ' 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
```

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

MAPI stores all custom properties that represent date and time information using Greenwich Mean Time (GMT). The OLE Messaging Library converts these properties so that the values appear to the user in local time.

For more information on the Field object's data types, see its **Type** property.

See Also

Creating and Sending a Message

Deleting a Message

The Message object's **Delete** method deletes the message.

To delete a message

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 after deleting it. Doing so can produce unpredictable results.

See Also

Searching for a Message

Filtering Messages in a Folder

A program sometimes needs to traverse an entire collection in order to take some action on all its members, such as displaying, sending, or copying them. But traversing a large collection like AddressEntries or Messages can take an inordinate amount of time. If you are only interested in certain members of the collection, your code can make efficient use of a filter.

The purpose of filtering is to limit the members of a collection that are presented to a traversing operation such as the Visual Basic **For Each** construction or a **GetFirst** … **GetNext** loop. The members are limited based on the values of the properties that you specify for the filter. Only those members that satisfy every filter property you have set are passed to your loop for processing.

In the case of messages in a folder, the hierarchy of objects is as follows:

Session object
    Folder object (Inbox or Outbox)
        Messages collection
            Message object
                Attachments collection
                Fields collection
                Recipients collection
            MessageFilter object
                Fields collection
                    Field object

Suppose, for example, you wish to find all unread messages received before a certain date, and to display the subject of each one. Before your display loop, you can set the message filter to limit the messages your loop sees. To do this, you obtain the Inbox folder, the folder's Messages collection, and the collection's MessageFilter. Next you set the filter's **Unread** property to **True** and its **TimeLast** property to the desired date. Then your loop deals only with the messages it needs.

This code fragment displays the **Subject** property of every message in the Inbox received before September 3, 1996 that has never been read:

```
Dim objSess, objInbox, objMsgColl, objMsgFilter As Object
Dim objMess As Message ' individual message processed in loop
On Error GoTo error_olemsg
Set objSess = CreateObject ( "MAPI.Session" )
objSess.Logon ' assume valid session for this example
Set objInbox = objSess.Inbox
If objInbox Is Nothing Then
    MsgBox "Invalid IPM Inbox from session"
    Exit Function
End If
Set objMsgColl = objInbox.Messages ' get Inbox's messages collection
' ( ... then validate the messages collection before proceeding ... )
Set objMsgFilter = objMsgColl.Filter
' ( ... then validate the message filter before proceeding ... )
objMsgFilter.TimeLast = DateValue ( "09/03/96" )
objMsgFilter.Unread = True ' filter for unread messages
' Message filter is now specified; ready for display loop
For Each objMess in objMsgColl ' performs loop, Sets each objMess
    MsgBox "Message not read: " & objMess.Subject
Next ' even easier than objMsgColl.GetFirst and .GetNext
error_olemsg:
MsgBox "Error " & Str(Err) & ": " & Error$(Err)
```

```
Exit Function ' so many steps to succeed; just exit on error
```

Handling Errors

The OLE Messaging Library raises exceptions for all errors. When you write Visual Basic applications that use the OLE Messaging Library, use the same run-time error handling techniques that you use in all your Visual Basic applications: the Visual Basic **On Error GoTo** statement.

Note that the error values and error handling techniques vary slightly depending on whether you are using Visual Basic version 4.0 or older versions of Visual Basic for Applications.

When you use older versions of Visual Basic for Applications, use the **Err** function to obtain the status code and the **Error$** function to obtain a descriptive error message, as in the following code fragment:

```
' Visual Basic for Applications error handling
MsgBox "Error number " & Err & " description. " & Error$(Err)
```

When you use Visual Basic 4.0, use the **Err** object's **Number** property to obtain the status code and its **Description** property to obtain the error message, as in the following fragment:

```
'' Visual Basic version 4.0 error handling
MsgBox "Error " & Err.Number & " description. " & Err.Description
```

Depending on your version of Microsoft Visual Basic, the error code is 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.

When you use Visual Basic 4.0, the error value is returned as the value of the MAPI **HRESULT** data type, a long integer error code. When you use Visual Basic for Applications, the error value is returned as the sum of decimal 1000 and the low-order word of **HRESULT**. This is because Visual Basic 3.0 reserves all run-time error values below 1000 for its own errors.

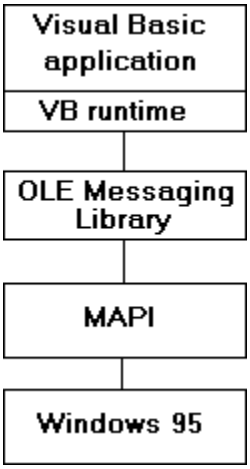
This code fragments checks for an error corresponding to the MAPI error code **mapiE_USER_CANCEL**, which has the value &H80040113. Visual Basic 4.0 users can check directly for this value. Visual Basic for Applications users check for the value of the low-order word plus decimal 1000. The low-order word is 0x0113, or 275, so the value returned by Visual Basic for Applications is 1275.

```
' demonstrates error handling for Logon
' Function: TestDrv_Util_CreateSessionAndLogon
' Purpose: Call the utility function Util_CreateSessionAndLogon
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
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: If Err.Number = mapiE_USER_CANCEL Then
    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
```

When an error occurs in the MAPI subsystem, the OLE Messaging Library supplies the error value returned by MAPI. However, the value can be returned from any of several 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 OLE Messaging Library at the next lower level. The OLE Messaging Library interacts with the MAPI system software, and the MAPI system software interacts with a lower layer of software, the operating system.



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 in this diagram, the Visual Basic application.

It is often useful to provide a general error handling capability that will display the complete **HRESULT** or error code value returned by the OLE Messaging Library.

For more information about run-time error handling and the **Err** object, see your product's Visual Basic documentation. For a listing of OLE Messaging Library and MAPI error values, see Error Codes.

See Also

Starting an OLE Messaging Session

Improving Application Performance

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

To access OLE Messaging Library objects, you create Visual Basic statements that concatenate the object names in sequence from left to right, separating objects with 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 period that appears in the statement. For example, the portion of the statement that says objSession.Inbox directs the OLE Messaging Library to create an internal Folder object that represents the user's Inbox. The next portion, .Messages, directs the OLE Messaging Library to create an internal Messages collection object. The final part, .GetFirst, directs the OLE Messaging Library to create an internal Message object that represents the first message in the user's Inbox. The statement contains three periods; the OLE Messaging Library creates three internal objects.

The best rule of thumb is to remember that periods 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.GetFirst.Text
MsgBox "Subj: " & objSession.Inbox.Messages.GetFirst.Subject
```

While this code generates correct results, it is not 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 is to generate the internal objects only once:

```
With objSession.Inbox.Messages.GetFirst
    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 code fragment is very efficient when your application reuses the Folder or Message objects or the Messages collection:

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

Now that you understand that a period in a statement directs the OLE Messaging Library to create a new internal object, you can see that the following two lines of code are not only not optimal but actually incorrect:

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

```
MsgBox("next message: " & 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 the OLE Messaging Library 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 therefore causes it to return its first message again.

Use the Visual Basic **With** statement or explicit variables to generate the expected results. The following code fragment shows both approaches:

```
' Use of the Visual Basic With statement
With objSession.Inbox.Messages
    Set objMessage = .GetFirst
    ' ...
    Set objMessage = .GetNext
End With
' Use of 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.

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.

Delivery and read notification may not be supported by all messaging systems.

See Also

Securing Messages

Moving a Message to Another Folder

The procedure documented in this section demonstrates, first, the old way to move message properties using the Messages collection's **Add** method and the Message object's **Delete** method, and then how to take advantage of the newer **MoveTo** method of the Message object.

Note With OLE Messaging Library version 1.0, the Message object's **Sender** property and other read-only properties of the Message object were not preserved during the first part of the procedure in this section. To preserve these properties using the old procedure, you had to append their text fields to read/write properties, such as the Message object's **Text** property.

With the **MoveTo** method, every property that is set on a Message object is automatically moved to the new Message object, regardless of whether it has read-only or read/write access. The access of every property is also preserved across the copy.

To move a message from one folder to another

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

- or -

2. Call the source Message object's **MoveTo** method.
3. Call the new Message object's **Update** method to save all new information in the MAPI system.
4. (Only necessary if you used the old **Add** and copy procedure) Call the source message's **Delete** method to delete the original message from its folder.

For more details on this procedure and a sample code fragment, see Copying a Message to Another Folder. The comment lines at the end of the first copy procedure contain the call to delete the original message:

```
' If MOVING a message to another folder, delete the original message:
objThisMsg.Delete
' Move operation implies that the original message is removed
```

This **Delete** call is not necessary if the **MoveTo** method is used.

Posting Messages to a Public Folder

To post a message to a public folder, create a message within the public folder by adding it to the folder's Messages collection. Then add your subject and message text as you would for other messages.

Note that for messages in public folders, you must also 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 **Sent**, **Submitted**, and **Unread** properties to **True**, and the **TimeReceived** and **TimeSent** properties to the current time.

When you are ready to make the message available, call the **Send** or **Update** method.

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

For more information on sending messages, see Creating And Sending A Message.

To create a message within a public folder

1. Call the Messages collection's **Add** method to create a Message object.
2. Set the Message object's **ConversationIndex**, **ConversationTopic**, **Subject**, **Text**, **TimeReceived**, **TimeSent**, and other message properties as desired.
3. Set the Message object's **Sent**, **Submitted**, and **Unread** 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
Function Util_NewConversation()
Dim objRecipColl As Recipients
Dim i As Integer
Dim objNewMsg As Message ' new message object
Dim strNewIndex As String
On Error GoTo error_olemsg

' objPublicFolder is a global variable that indicates
' the folder in which you want to post the message
Set objNewMsg = objPublicFolder.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"
    .Text = "Wanted: Apollo or Mercury spacecraft with low mileage."
    .ConversationTopic = .Subject
```

```
    .ConversationIndex = Util_GetEightByteTimeStamp() ' utility
    .TimeReceived = Time
    .TimeSent = .TimeReceived
    .Sent = True
    .Submitted = True
    .Unread = True
    .Update
    .Send showDialog:=False
End With
Exit Function

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

End Function
```

For more information on the **ConversationIndex** property, see Working With Conversations.

See Also

Searching for a Folder

Reading a Message from the Inbox

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

As described in OLE Messaging Library Object Design, the OLE Messaging Library objects are organized in a hierarchy. The Session object at the topmost level allows access to a Folder object. Each folder contains a Messages collection, which contains individual Message objects. The text of the message appears in its **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's **Messages** property to obtain a Messages collection object, and calls the collection's methods to get a specific message.

This code fragment assumes that the application has already created the Session object variable *objSession* and successfully called the Session object's **Logon** method, as described in Starting an OLE Messaging Session:

```
Dim objSession As Session     ' Session object
Dim objInboxFolder As Folder  ' Folder object
Dim objInMessages As Messages ' Messages collection
Dim objOneMsg As Message      ' 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 attempt to set an object variable without using the **Set** keyword, Visual Basic generates an error message.

The preceding code fragment declares several object variables. However, it is also possible to access the message with fewer variables. The following code fragment is equivalent to the preceding code, and is preferable if you have no subsequent need for the Inbox folder or its Messages collection:

```
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 Improving Application Performance.

See Also

Creating and Sending a Message, 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 navigate through the Folders collection. Search for a specific folder by comparing each folder's properties with the desired properties.

Using the Session Object's 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** property is a string representation of the MAPI entry identifier and its value is determined by the service provider.

The following code fragment contains code that saves the identifier 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 navigate through the collection, examining properties of each Folder object to determine whether it is the folder you want.

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

The following code fragment 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 ' went off end of loop
        Util_GetFolderByName = True ' success
    End If
    Exit Function

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

You can also navigate upward through the folder hierarchy by using each Folder object's **Parent** property.

See Also

Searching for a Message

Searching for a Message

To access a message, you can search for it 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 navigate through the folder's Messages collection. Search for a specific message by comparing the current Message object's properties with the desired properties.
- Obtain a MessageFilter object from the Filter property of the Messages collection. Set the desired properties for filtering, and then use the **Get** methods, which return only the messages matching the filter settings.

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

The following code fragment contains code that saves the identifier for the message, 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 navigate through the collection, examining properties of each Message object to determine if it is the message you want.

The OLE Messaging Library supports the **GetFirst**, **GetNext**, **GetLast**, and **GetPrevious** methods for the Messages collection object. You can also use the Visual Basic **For Each** construction to traverse the collection.

Note that, with the OLE Messaging Library version 1.1 and later, you can use a MessageFilter object to restrict a search with the **Get** methods. Obtain the message filter through the Messages collection's Filter property, set the filter's properties to the values desired for the search, and then proceed with the **Get** methods. Only the messages passing the filter criteria are returned for your inspection. For more information on message filtering, see Filtering Messages in a Folder.

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 Message ' 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 Folder

Securing Messages

A 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 one or both of these flags to **True**.

These flags simply represent a request to the underlying messaging service. Whether the message gets encrypted or digitally signed depends on whether these security measures are implemented by your messaging service.

Neither MAPI nor the OLE Messaging Library performs encryption or digital signing. The OLE Messaging Library simply sets the appropriate 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 server administrator.

```
Dim objMessage As Message ' assume valid Message object
' ...
objMessage.Encrypted = True ' can also set objMessage.Signed = 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 on to 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 (GAL) or your personal address book (PAB).

As described in OLE Messaging Library Object Design, the OLE Messaging Library 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, no type)
            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 fragment uses the Recipients collection returned by the Session object's **AddressBook** method.

This code fragment assumes that the application has already created the Session object variable *objSession* and successfully called the Session object's **Logon** method, as described in Starting an OLE Messaging Session:

```
' 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: first parameter ("recipients") not used in this call
    ' recipients:=objInitRecipColl initializes recipients for dialog
    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 an OLE Messaging Session

As described in OLE Messaging Library 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: If Err.Number = mapiE_USER_CANCEL Then
    MsgBox "User pressed Cancel"
Else
    MsgBox "Unrecoverable Error:" & Err
End If
Util_CreateSessionAndLogon = False
Exit Function

End Function
```

The way you deal with errors depends on your version of Visual Basic. For more information, see Handling Errors.

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
    objSession.Logon profileName:=strProfileName, _
                     showDialog:=False
End If
Exit Function
```

```
error_olemsg:
If 1273 = Err Then ' VB4.0: If Err.Number = mapiE_LOGON_FAILED 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 when a user cancels from the Logon dialog box. For more information, see Handling Errors. For a listing of OLE Messaging Library and MAPI error values, see Error Codes.

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 Reading a Message from the Inbox.

See Also

Creating and Sending a Message

Using Addresses

In general, 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 supports both kinds of addresses with its Recipient object. To look up an address for a name, you supply the **Name** property only. To use custom addresses, you supply the full address in the **Address** property.

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" could be mapped to the e-mail address "johnd".

In contrast to 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 the corresponding address. When the display name is ambiguous, or can match more than one entry in the address book, the MAPI system prompts the user to select from a list of possible matching names.

The Recipient object's **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 object's **AddressEntry** property. The Recipient and AddressEntry object properties are related as follows:

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

The Recipient object's **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 information that appears in the AddressEntry **Address** and **Type** properties.

You can also supply a complete recipient address. 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

object's **Address** property must use the following syntax:

AddressType:*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 the 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 its **Resolve** method. Note that the **Resolve** method can display a dialog box.
- 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 earlier, with the colon character (:) separating the address type from the address, and call the **Resolve** method.

The following code fragment 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 Message   ' new message to add recipients to
Dim objNewRecips As Recipients ' recipients of new message
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
Set objNewRecips = objNewMessage.Recipients

' Add three recipients
' 1. look up entry in address book specified by profile
Set objOneRecip = objNewRecips.Add( _
```

```
                                Name:=strName, _
                                Type:=mapiTo)
If objOneRecip Is Nothing Then
    MsgBox "Unable to add recipient using display name"
    Exit Function
End If
objOneRecip.Resolve ' this looks up the entry

' 2. add a custom recipient
Set objOneRecip = objNewRecips.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
Set objOneRecip = objNewRecips.Add( _
                                entryID:=strAddrEntryID, _
                                Name:=strName, _
                                Type:=mapiTo)
If objOneRecip Is Nothing Then
    MsgBox "Unable to add recipient using existing address entry"
    Exit Function
End If

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

objNewMessage.Subject = "Addressing test"
objNewMessage.Update ' commit the message to storage in MAPI system
objNewMessage.Send(showDialog:=False)
Exit Function

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

End Function
```

See Also

Changing an Existing Address Entry

Viewing MAPI Properties

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

The Fields collection's **Item** property allows you to specify the actual *property tag* value as an identifier. The MAPI 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.

Note You can only use the MAPI property tag on 32-bit platforms. This method of access is not available on any other platform.

The OLE Messaging Library also supports *multivalued* properties, or properties that represent arrays of values. A multivalued property appears to the Visual Basic application as a variant array. You can use the **For ... Next** construction or **For Each** statement to access individual array entries.

Note Do not mix data types within an OLE variant array that you are going to use with the OLE Messaging Library. Unlike variant array members, every member of a MAPI multivalued property must be of the same type. Setting mixed types in a variant array and presenting it to MAPI as a multivalued property results in MAPI errors.

The OLE Messaging Library works with three types of message properties:

- Standard MAPI properties with property tags defined as constants by the OLE Messaging Library, such as **mapiPR_MESSAGE_CLASS**.
- Standard MAPI properties not defined by the OLE Messaging Library. The Object Browser can tell you if the property you want to access is defined.
- Custom properties created and named by the application.

The Fields collection collection exposes standard MAPI properties not defined by the OLE Messaging Library and custom properties created and named by the application. The **Item** property selects an individual Field object either by its MAPI property tag or by its custom name.

Although the Field object provides a **Delete** method, some standard MAPI properties, such as those created by MAPI system components, cannot be deleted.

MAPI stores all properties that represent date and time information using Greenwich Mean Time (GMT). The OLE Messaging Library converts these properties so that the values appear to the user in local time.

For definitions and details on all standard MAPI properties, 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: 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 = &H001A001E ' VB4.0: lValue = mapiPR_MESSAGE_CLASS
' &H001A = PR_MESSAGE_CLASS; &H001E = PT_TSTRING
' high-order 16 bits = property ID, low-order = 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

Customizing a Folder or Message

Working with Conversations

Two Message object properties let you show 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 Message object properties **ConversationIndex** and **ConversationTopic** give you an easy way to organize and display messages. Rather than simply grouping messages by subject, time received, or sender, you can show conversational relationships among messages. The **ConversationTopic** property is a string that describes the overall subject of the conversation. All messages within the same conversation use the same value for the **ConversationTopic** property. The **ConversationIndex** property is a hexadecimal string that you can use to represent the relationships between the messages in the thread. Each message in the conversation should have a different **ConversationIndex** property.

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's **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. This convention uses concatenated time stamp values. The first time stamp in the **ConversationIndex** string represents the original message. Whenever a message replies to a conversation message, it 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. Using this convention, you can easily see relationships among messages when you sort the messages by **ConversationIndex** values.

The following code fragment provides a utility function, **Util_GetEightByteTimeStamp**, which can be used to build Microsoft Exchange Server compatible **ConversationIndex** values. The utility function calls the OLE function **CoCreateGuid** to obtain the time stamp value from a **GUID** 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, as follows:

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

For additional sample code dealing with conversations, see Posting Messages to a Public Folder.

Objects, Properties, and Methods

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

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

Object	Available in version	Properties	Methods
AddressEntries collection	1.1	Application, Class, Count, Filter, Item, Parent, Session	Add, Delete, GetFirst, GetLast, GetNext, GetPrevious, Sort
AddressEntry	1.0.a	Address, Application, Class, DisplayType, Fields, ID, Members, Name, Parent, Session, Type	Delete, Details, IsSameAs, Update
AddressEntryFilter	1.1	Address, Application, Class, Fields, Name, Not, Or, Parent, Session	IsSameAs
AddressList	1.1	AddressEntries, Application, Class, ID, Index, IsReadOnly, Name, Parent, Session	IsSameAs
AddressLists collection	1.1	Application, Class, Count, Item, Parent, Session	(none)
Attachment	1.0.a	Application, Class, Fields, Index, Name, Parent, Position, Session, Source, Type	Delete, IsSameAs, ReadFromFile, WriteToFile
Attachments collection	1.0.a	Application, Class, Count, Item, Parent, Session	Add, Delete
Field	1.0.a	Application, Class, ID, Index, Name, Parent, Session, Type, Value	Delete, ReadFromFile, WriteToFile
Fields collection	1.0.a	Application, Class, Count, Item, Parent, Session	Add, Delete, SetNamespace
Folder	1.0.a	Application, Class, Fields, FolderID, Folders, ID, MAPIOBJECT[1],	CopyTo, Delete, IsSameAs, MoveTo, Update

		Messages, Name, Parent, Session, StoreID	
Folders collection	1.0.a	Application, Class, Count, Item, Parent, Session	Add, Delete, GetFirst, GetLast, GetNext, GetPrevious, Sort
InfoStore	1.0.a	Application, Class, ID, Index, Name, Parent, ProviderName, RootFolder, Session	IsSameAs
InfoStores collection	1.0.a	Application, Class, Count, Item, Parent, Session	(none)
Message	1.0.a	Application, Attachments, Class, Conversation, 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	CopyTo, Delete, IsSameAs, MoveTo, Options, Send, Update
MessageFilter	1.1	Application, Class, Conversation, Fields, Importance, Not, Or, Parent, Recipients,	IsSameAs

		Sender, Sent, Session, Size, Sort, Subject, Text, TimeFirst, TimeLast, Type, Unread	
Messages collection	1.0.a	Application, Class, Count, Filter, Item, Parent, Session	Add, Delete, GetFirst, GetLast, GetNext, GetPrevious, Sort
Recipient	1.0.a	Address, AddressEntry, Application, Class, DisplayType, ID, Index, Name, Parent, Session, Type	Delete, IsSameAs Resolve
Recipients collection	1.0.a	Application, Class, Count, Item, Parent, Resolved, Session	Add, Delete, Resolve
Session	1.0.a	AddressLists, Application, Class, CurrentUser, Inbox, InfoStores, MAPIOBJECT[1], Name, OperatingSystem, Outbox, Parent, Session, Version	AddressBook, CompareIDs, DeliverNow, GetAddressEntry, GetFolder, GetInfoStore, GetMessage, Logoff, Logon

1 The MAPIOBJECT property is not available to Visual Basic applications. For more information, see the reference for the MAPIOBJECT property.

This reference is organized by object. For each object there is a summary topic, followed by reference documentation for each property or method that belongs to the object. The properties and methods are organized alphabetically.

Each property or method topic in the reference displays a **Group** button following the topic title. Clicking this button displays the summary topic for the object to which the property or method belongs. The summary topic includes tables of the object's properties and methods.

To avoid duplication, the section Properties Common to All OLE Messaging Library Objects describes the properties that have the same meaning for all OLE Messaging Library objects. These are:

- **Application**
- **Class**
- **Parent**
- **Session**

Object Model

The object model for the OLE Messaging Library is a hierarchical model. In the following table, each indented object is considered a child of the object under which it is indented. An object is the parent of every object at the next level of indentation under it. For example, an Attachments collection and a Recipients collection are both child objects of a Message object, and a Messages collection is a parent object of a Message object. However, a Messages collection is not a parent object of a Recipients collection.

Session
- AddressLists collection
  - AddressList
    - AddressEntries collection
      - AddressEntry
        - Fields collection
          - Field
      - AddressEntryFilter
        - Fields collection
          - Field
- Folder (Inbox or Outbox)
  - Fields collection
    - Field
  - Folders collection
    - Folder
      - … Folder …
  - Messages collection
    - Message
      - Attachments collection
        - Attachment
      - Fields collection
        - Field
      - Recipients collection
        - Recipient
          - AddressEntry
            - Fields collection
              - Field
    - MessageFilter
      - Fields collection
        - Field
- InfoStores collection
  - InfoStore
    - Folder
      - … Folder …

The notation "… Folder …" signifies that a folder can contain other folders, which in turn can contain more folders, nested to an indefinite level.

Properties Common to All OLE Messaging Library 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 immediate parent of the object, and the **Class** property is an integer value that identifies the OLE Messaging Library object.

All four of these common properties have read-only access in all objects. 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 Library object hierarchy and has no parent.

To reduce duplication, the detailed reference for these properties appears only once, in this section. The following table lists the properties that are common to all OLE Messaging Library objects and that have the same meaning for all objects.

Properties

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

Application Property Group

The **Application** property returns the name of the active application, namely the OLE Messaging Library. Read-only.

Syntax

object.**Application**

Data Type

String

Remarks

The **Application** property always contains the string “OLE/Messaging”.

By always returning the same string, Microsoft OLE Messaging Library differs from other implementations of Automation servers. Many Automation servers are based on executable files, which take the extension .EXE and return an object value. The OLE Messaging Library, being part of the MAPI subsystem, is implemented with dynamic-link libraries, which take the extension .DLL.

As of version 1.1, the OLE Messaging Library is an in-process server, residing in a .DLL file and linking directly with the calling modules. In comparison with the former local server architecture, this removes the need for RPCs across process boundaries and greatly improves the performance of OLE Messaging Library calls.

The version number of the OLE Messaging Library is available through the Session object’s **Version** property.

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
```

Class Property Group

The **Class** property returns the object class of the object. Read-only.

Syntax

object.**Class**

Data Type

Long

Remarks

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
AddressEntries collection	21	**mapiAddressEntries**
AddressEntry	8	**mapiAddressEntry**
AddressEntryFilter	9	**mapiAddressFilter**
AddressList	7	**mapiAddressList**
AddressLists collection	20	**mapiAddressLists**
Attachment	5	**mapiAttachment**
Attachments collection	18	**mapiAttachments**
Field	6	**mapiField**
Fields collection	19	**mapiFields**
Folder	2	**mapiFolder**
Folders collection	15	**mapiFolders**
InfoStore	1	**mapiInfoStore**
InfoStores collection	14	**mapiInfoStores**
Message	3	**mapiMsg**
MessageFilter	10	**mapiMessageFilter**
Messages collection	16	**mapiMessages**
Recipient	4	**mapiRecipient**
Recipients collection	17	**mapiRecipients**
Session	0	**mapiSession**

The OLE Messaging Library also defines **mapiUnknown**, with value -1, for an object implementing the OLE **IUnknown** interface.

Example

```
' Function: Util_DecodeObjectClass
```

```
' Purpose: Decode the long integer class value,
'          show the related object name
' See documentation topic: Class property
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
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 = objMessages.GetFirst
If objOneMsg Is Nothing Then
    MsgBox "Inbox is empty"
    Exit Function
End If
' expect type mapiMessage = 3 for Message object
Util_DecodeObjectClass (objOneMsg.Class)
' error handling here ...
End Function
```

Parent Property

The **Parent** property returns the parent of the object. Read-only.

Syntax

Set *objParent* = *object*.**Parent**

Data Type

Object

Remarks

The **Parent** property in the OLE Messaging Library returns the *immediate* parent of an object. The immediate parent for each object is shown in the following table.

OLE Messaging Library object	Immediate parent in object hierarchy
AddressEntries collection	AddressList
AddressEntry (returned by Session.**CurrentUser**)	AddressEntries collection
AddressEntry (all others)	Recipient
AddressEntryFilter	AddressEntries collection
AddressList	AddressLists collection
AddressLists collection	Session
Attachment	Attachments collection
Attachments collection	Message
Field	Fields collection
Fields collection	AddressEntry, AddressEntryFilter, Folder (Inbox or Outbox), Message, or MessageFilter
Folder (Inbox or Outbox)	Session
Folder (all others)	Folders collection or InfoStore
Folders collection	Folder (Inbox or Outbox)
InfoStore	InfoStores collection
InfoStores collection	Session
Message	Messages collection
MessageFilter	Messages collection
Messages collection	Folder (Inbox or Outbox)
Recipient	Recipients collection
Recipients collection	Message
Session	Set to **Nothing**

The **Parent** property represents the *immediate* parent of the object, rather than the *logical* parent. For example, a folder contains a Messages collection, which contains Message objects. The **Parent** property for a message is the immediate parent, the Messages collection, rather than the logical parent, the Folder object.

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

For more information on the OLE Messaging Library object hierarchy, see Object Model.

Example

This code fragment displays the **Class** of the parent Messages collection of a Message object:

```
' 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 object, you have to take the parent of the Messages collection:

```
' Function: Messages_Parent
' Purpose: Display the Messages collection Parent class value
' See documentation topic: Parent property
Function Messages_Parent()
Set objMessages = objOneMsg.Parent
' error handling here ...
If objMessages Is Nothing Then
    MsgBox "No active Messages collection"
    Exit Function
End If
MsgBox "Messages collection parent class = " & objMessages.Parent.Class
Exit Function
' error handling here ...
End Function
```

Session Property

The **Session** property returns the top-level Session object associated with the specified OLE Messaging Library object. Read-only.

Syntax

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

Data Type

Object

Remarks

The Session object represents the highest level in the OLE Messaging Library 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
Function Folder_Session()
Dim objSession2 As Session ' 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's Name = " & objSession2.Name
Set objSession2 = Nothing
' error handling here ...
End Function
```

AddressEntries Collection Object

The AddressEntries collection object contains one or more AddressEntry objects.

Quick Info

Specified in header file:	MDISP.ODL
First available in:	OLE Messaging Library version 1.1
Parent objects:	AddressList
Child objects:	AddressEntry AddressEntryFilter
Default property:	Item

An AddressEntries collection is considered a *large collection*, which means that the **Count** and **Item** properties have limited validity, and your best option is to use an AddressEntry object identifier value or the **Get** methods to access an individual AddressEntry object within the collection.

Properties

Name	Available in version	Type	Access
Application	1.1	String	Read-only
Class	1.1	Long	Read-only
Count	1.1	Long	Read-only
Filter	1.1	AddressEntryFilter object	Read/write
Item	1.1	AddressEntry object	Read-only
Parent	1.1	AddressList object	Read-only
Session	1.1	Session object	Read-only

Methods

Name	Available in version	Parameters
Add	1.1	*emailtype* as **String**, (optional) *name* as **String**, (optional) *address* as **String**
Delete	1.1	(none)
GetFirst	1.1	(none)
GetLast	1.1	(none)
GetNext	1.1	(none)
GetPrevious	1.1	(none)
Sort	1.1	(optional) *SortOrder* as **Long**, (optional) *PropTag* as **Long**, (optional) *PropID* as **String**

Remarks

Each AddressEntry object in the collection holds information representing a person or process to which the messaging system can deliver messages. An AddressEntries collection provides access to the entries in a MAPI address book.

Large collections, such as the AddressEntries collection, cannot always maintain an accurate count of the number of objects in the collection. It is strongly recommended that you use the **GetFirst**, **GetNext**, **GetLast**, and **GetPrevious** methods to access individual items in the collection. You can access one specific address entry by using the Session object's **GetAddressEntry** method, and you can access all the items in the collection with the Visual Basic **For Each** construction.

The order that items are returned by **GetFirst**, **GetNext**, **GetLast**, and **GetPrevious** depends on whether the address entries are sorted or not. The AddressEntry objects within a collection can be sorted on a MAPI property of your choice, either ascending or descending, using the **Sort** method. When the items are not sorted, you should not rely on these methods to return the items in any specified order. The best programming approach to use with unsorted collections is to assume that the access functions are able to access all items within the collection, but that the order of the objects is not defined.

Add Method (AddressEntries Collection)

The **Add** method creates and returns a new AddressEntry object in the AddressEntries collection.

Syntax

Set *objAddressEntry* = *objAddrEntriesColl*.**Add(***emailtype* **[**, *name, address* **])**

Parameters

objAddressEntry
On successful return, contains the new AddressEntry object.

objAddrEntriesColl
Required. The AddressEntries collection object.

emailtype
Required. String. The address type of the address entry.

name
Optional. String. The display name or alias of the address entry.

address
Optional. String. The full messaging address of the address entry.

Remarks

The *emailtype* parameter corresponds to the PR_ADDRTYPE property and qualifies the *address* parameter by specifying which messaging system the address is valid in. Typical values are SMTP, FAX, and X400.

The *emailtype*, *name*, and *address* parameters correspond to the **Type**, **Name**, and **Address** properties of the AddressEntry object.

The **DisplayType** property of the new AddressEntry object is set to **mapiUser**, indicating a local messaging user has been created. The **DisplayType** property is read-only and cannot subsequently be changed.

The user must have permission to **Add** or **Delete** an AddressEntry object. Most users have this permission only for their personal address book (PAB).

The new AddressEntry object is saved in the MAPI system when you call its **Update** method.

Example

This code fragment adds a new entry to a user's personal address book (PAB). Note the use of the **Item** property as the default property of both the AddressLists and Fields collections.

```
' get PAB AddressList from AddressLists collection of Session
Set myList = MAPI.Session.AddressLists("Personal Address Book")
' add new AddressEntry to AddressEntries collection of AddressList
Set newEntry = myList.AddressEntries.Add "FAX", "John Doe"
' add FaxNumber field to new AddressEntry and give it a value
newEntry.Fields.Add "FaxNumber", vbString
newEntry.Fields("FaxNumber") = "+1-206-555-7069"
' commit new entry, field, and value to PAB AddressList
newEntry.Update
```

Count Property (AddressEntries Collection)

The **Count** property returns the number of AddressEntry objects in the collection, or a very large number if the exact count is not available. Read-only.

Syntax

objAddrEntriesColl.**Count**

Data Type

Long

Remarks

The **Count** property is useful for determining whether an AddressEntries collection is empty or not.

A large collection cannot always maintain an accurate count of its members, and the **Count** property cannot be used as the collection's size when it has a very large value such as **mapiMaxCount**. Programmers needing to access individual objects in a large collection are strongly advised to use the Visual Basic **For Each** statement or the **Get** methods.

The recommended procedures for traversing a large collection are, in decreasing order of preference:

1. Global selection, such as the Visual Basic **For Each** statement.
2. The **Get** methods, particularly **GetFirst** and **GetNext**
3. An indexed loop, such as the Visual Basic **For ... Next** construction.

If the address book provider cannot supply the precise number of AddressEntry objects, the OLE Messaging Library returns a very large number for the **Count** property. On 32-bit platforms, this value is **mapiMaxCount**, which equals 2^31 - 1, or 2147483647. On other platforms, **mapiMaxCount** is not defined, and the OLE Messaging Library returns -1. A program on such a platform must ensure that -1 does not prematurely terminate any iteration based on the **Count** property.

Programmers using an indexed loop terminating on the **Count** property must also check each returned object for a value of **Nothing**. The loop must proceed forward from the beginning of the collection, and the index must have initial and increment values of 1. Results are undefined for any other procedure.

The use of the **Item** property in conjunction with the **Count** property in a large collection can be seen in the following example.

Example

This code fragment counts the AddressEntry objects in a user's personal address book (PAB):

```
Dim i As Integer                ' loop index / object counter
Dim myPAB as AddressList        ' personal address book AddressList
Dim myPABColl as AddressEntries ' AddressEntries collection of PAB
' select PAB from AddressLists collection of Session
Set myPAB = MAPI.Session.AddressLists.Item("Personal Address Book")
' .Item could have been omitted above since it is default property
' make sure returned AddressList object is valid
If myPAB Is Nothing Then
    ' MsgBox "PAB object is invalid"
    ' Exit
End If
' get AddressEntries collection of PAB AddressList
Set myPABColl = myPAB.AddressEntries
' see if PAB is empty
i = myPABColl.Count ' valid if not a "very large number"
If 0 = i Then ' collection empty; 0 is correct count
```

```
    MsgBox "No AddressEntry items in PAB"
ElseIf mapiMaxCount = i Then ' .Count is not valid; get exact count
    For i = 0 To myPABColl.Count Step 1
        If myPABColl.Item(i) Is Nothing Then
            Exit For ' end of collection; members are 0, ... , i - 1
        End If
    Next i
End If
```

Delete Method (AddressEntries Collection)

The **Delete** method deletes all the address entries in the AddressEntries collection.

Syntax

objAddrEntriesColl.**Delete()**

Parameters

objAddrEntriesColl
Required. The AddressEntries collection object.

Remarks

The **Delete** operation invalidates all the AddressEntry objects in the collection but does not remove them from memory. The programmer should set the invalidated objects to **Nothing** to remove them from memory, or reassign them to other address entries. Attempted access to a deleted object results in a return of **mapiE_INVALID_OBJECT**.

Be cautious using the **Delete** method with a collection, because it deletes all the collection's member objects. To delete only one AddressEntry object, use the **Delete** method specific to that object.

The **Delete** method on a large collection takes effect immediately and is permanent. A deleted member cannot be recovered. However, the collection itself it still valid, and you can **Add** new members to it.

Filter Property (AddressEntries Collection)

The **Filter** property returns an AddressEntryFilter object for the AddressEntries collection. Read/write.

Syntax

objAddrEntriesColl.**Filter**

Data Type

Object

Remarks

An AddressEntryFilter object with no criteria is created by default for every AddressEntries collection. When you specify criteria by setting properties in the filter's Fields collection, the filter restricts any subsequent search on the AddressEntries collection. For more information, see the AddressEntryFilter Object and Filtering Messages in a Folder.

GetFirst Method (AddressEntries Collection)

The **GetFirst** method returns the first AddressEntry object in the AddressEntries collection. It returns **Nothing** if no first object exists.

Syntax

Set *objAddressEntry* = *objAddrEntriesColl*.**GetFirst()**

Parameters

objAddressEntry

On successful return, represents the first AddressEntry object in the collection.

objAddrEntriesColl

Required. The AddressEntries collection object.

Remarks

The **Get** methods are similar to the **Find** and **Move** methods used with Microsoft Access, but they use a different syntax.

GetLast Method (AddressEntries Collection)

The **GetLast** method returns the last AddressEntry object in the AddressEntries collection. It returns **Nothing** if no last object exists.

Syntax

Set *objAddressEntry* = *objAddrEntriesColl*.**GetLast()**

Parameters

objAddressEntry

On successful return, represents the last AddressEntry object in the collection.

objAddrEntriesColl

Required. The AddressEntries collection object.

Remarks

The **Get** methods are similar to the **Find** and **Move** methods used with Microsoft Access, but they use a different syntax.

GetNext Method (AddressEntries Collection)

The **GetNext** method returns the next AddressEntry object in the AddressEntries collection. It returns **Nothing** if no next object exists, for example if already positioned at the end of the collection.

Syntax

Set *objAddressEntry* = *objAddrEntriesColl*.**GetNext()**

Parameters

objAddressEntry

On successful return, represents the next AddressEntry object in the collection.

objAddrEntriesColl

Required. The AddressEntries collection object.

Remarks

The **Get** methods are similar to the **Find** and **Move** methods used with Microsoft Access, but they use a different syntax.

GetPrevious Method (AddressEntries Collection)

The **GetPrevious** method returns the previous AddressEntry object in the AddressEntries collection. It returns **Nothing** if no previous object exists, for example if already positioned at the beginning of the collection.

Syntax

Set *objAddressEntry* = *objAddrEntriesColl*.**GetPrevious()**

Parameters

objAddressEntry

On successful return, represents the previous AddressEntry object in the collection.

objAddrEntriesColl

Required. The AddressEntries collection object.

Remarks

The **Get** methods are similar to the **Find** and **Move** methods used with Microsoft Access, but they use a different syntax.

Item Property (AddressEntries Collection)

The **Item** property returns a single AddressEntry object from the AddressEntries collection. Read-only.

Syntax

objAddrEntriesColl.**Item(***index***)** or *objAddrEntriesColl***(***index***)**

objAddrEntriesColl.**Item(***prefix***)** or *objAddrEntriesColl***(***prefix***)**

index
A long integer ranging from 1 to the size of the AddressEntries collection.

prefix
A string representing a prefix substring of an AddressEntry object's **Name** property.

Data Type

Object

Remarks

The **Item** property is useful for satisfying syntax requirements when obtaining a member of an AddressEntries collection.

A large collection cannot support true integer indexing, and the **Item(***index***)** syntax cannot be used for arbitrary selection of members of the collection. Programmers needing to access individual objects in a large collection are strongly advised to use the Visual Basic **For Each** statement or the **Get** methods, particularly **GetFirst** and **GetNext**.

The **Item(***index***)** syntax is provided solely as a placeholder in an indexed loop, such as the **For ... Next** construction in Visual Basic. Such a loop must proceed forward from the beginning of the collection, and the index must have initial and increment values of 1. Results are undefined for any other procedure.

For more information on using the **Count** and **Item** properties in a large collection, see the example in the **Count** property.

The **Item(***prefix***)** syntax returns the first AddressEntry object whose **Name** property begins with the string specified by *prefix*.

The **Item** property is the default property of an AddressEntries collection, meaning that *objAddrEntriesColl***(***prefix***)** is syntactically equivalent to *objAddrEntriesColl*.**Item(***prefix***)** in Visual Basic code.

Sort Method (AddressEntries Collection)

The **Sort** method sorts the address entries in the collection on the specified property according to the specified sort order.

Syntax

objAddrEntriesColl.**Sort([***SortOrder, PropTag***])**

objAddrEntriesColl.**Sort([***SortOrder, PropID***])**

Parameters

objAddrEntriesColl

Required. The AddressEntries collection object.

SortOrder

Optional. Long. The specified sort order, one of the following values:

Value	Numeric value	Description
mapiNone	0	No sort
mapiAscending	1	Ascending sort (default)
mapiDescending	2	Descending sort

PropTag

Optional. Long. The property tag value for the MAPI property to be used for the sort. *PropTag* is the 32-bit MAPI property tag associated with the property, such as **mapiPR_EMAIL_ADDRESS**.

PropID

Optional. String. The custom property name of a MAPI named property.

Remarks

Both parameters are optional. If *SortOrder* is not specified, ascending order is used. If neither *PropTag* nor *PropID* is specified, the property used in the previous call to **Sort** is used again. If **Sort** has never been called on this collection during this session, the MAPI property **mapiPR_DISPLAY_NAME** is used for the sort.

AddressEntry Object

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

Quick Info

Specified in header file:	MDISP.ODL
First available in:	OLE Messaging Library version 1.0.a
Parent objects:	AddressEntries collection Recipient
Child objects:	Fields collection
Default property:	Name

When an AddressEntry object is used as a child object of a Recipient object, it 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

Name	Available in version	Type	Access
Address	1.0.a	String	Read/write
Application	1.0.a	String	Read-only
Class	1.0.a	Long	Read-only
DisplayType	1.0.a	Long	Read-only
Fields	1.0.a	Field object or Fields collection object	Read-only
ID	1.0.a	String	Read-only
Members	1.0.a	AddressEntries collection object	Read-only
Name	1.0.a	String	Read/write
Parent	1.0.a	AddressEntries collection object or Recipient object	Read-only
Session	1.0.a	Session object	Read-only
Type	1.0.a	String	Read/write

Methods

Name	Available in version	Parameters
Delete	1.0.a	(none)
Details	1.0.a	(optional) *parentWindow* as **Long**
IsSameAs	1.1	(required) *object* as **Object**
Update	1.0.a	(optional) *makePermanent* as **Boolean**,

(optional) *refreshObject* as **Boolean**

Address Property (AddressEntry Object)

The **Address** property specifies the messaging address of an address entry or message recipient. Read/write.

Syntax

objAddressEntry.**Address**

Data Type

String

Remarks

The AddressEntry object's **Address** property contains a unique string that identifies a message recipient and provides 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 can be combined to form the *full address*, the complete messaging address that appears in the Recipient object's **Address** property using the following syntax:

AddressType:*AddressValue*

The AddressEntry object's **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_Item()
If 0 = objRecipColl.Count Then
    MsgBox "No recipients in the list"
    Exit Function
End If
iRecipCollIndex = 1
Set objOneRecip = objRecipColl.Item(iRecipCollIndex)
' could also be objRecipColl(iRecipCollIndex) since .Item is default

' 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
```

Delete Method (AddressEntry Object)

The **Delete** method deletes the AddressEntry object.

Syntax

objAddressEntry.**Delete()**

Parameters

objAddressEntry
Required. The AddressEntry object.

Remarks

The action of the **Delete** method is permanent, and the AddressEntry object cannot be recovered. Before calling the **Delete** method, the application can prompt the user to verify whether the address entry should be permanently deleted.

The **Delete** operation invalidates the AddressEntry object but does not remove it from memory. The programmer should set the invalidated object to **Nothing** to remove it from memory, or reassign it to another address entry. Attempted access to a deleted object results in a return of **mapiE_INVALID_OBJECT**.

You can delete all the address entries in the AddressEntries collection by calling the collection's Delete method. The ability to delete any address entry depends on the permissions granted to the user. The **Delete** method returns an error code if called with insufficient permissions.

Example

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

See Also

Add Method (Recipients Collection)

Details Method (AddressEntry Object)

The **Details** method displays a modal 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) specifies that the dialog should be application-modal.

Remarks

The **Details** dialog is always modal, meaning the parent window is disabled while the dialog is active. If the *parentWindow* parameter is set to **0** or is not set, all windows belonging to the application are disabled while the dialog is active. If the *parentWindow* parameter is supplied but is not valid, the call returns **mapiE_INVALID_PARAMETER**.

The dialog box must always contain at least the display name and messaging address of the address entry. The **Details** method fails if either the **Name** or **Address** property is empty.

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

DisplayType Property (AddressEntry Object)

The **DisplayType** property returns the display type of the address entry. Read-only.

Syntax

objAddressEntry.**DisplayType**

Data Type

Long

Remarks

The **DisplayType** property enables special processing based on its value, such as displaying an associated icon. You can also use the display type to sort or filter address entries.

The following values are defined:

DisplayType value	Description
mapiUser	Local user
mapiDistList	Distribution list
mapiForum	Public folder
mapiAgent	Agent
mapiOrganization	Organization
mapiPrivateDistList	Private distribution list
mapiRemoteUser	Remote user

The **DisplayType** property is always set to **mapiUser** when you **Add** a new address entry to an AddressEntries collection. It cannot subsequently be changed.

Fields Property (AddressEntry Object)

The **Fields** property returns a single Field object or a Fields collection object. Read-only.

Syntax

objAddressEntry.**Fields**

objAddressEntry.**Fields(***index***)**

objAddressEntry.**Fields(***proptag***)**

objAddressEntry.**Fields(***name***)**

index
Short integer (less than or equal to 65535). Specifies the index within the collection.

proptag
Long integer (greater than or equal to 65536). Specifies the property tag value for the MAPI property to be retrieved.

name
String. Specifies the name of the custom MAPI property.

Data Type

Object

Remarks

The **Fields** property returns one or more of the fields associated with an AddressEntry object. Each field typically corresponds to a MAPI property.

The **Fields** property provides a generic access mechanism that allows Visual Basic and Visual C++ programmers to retrieve the value of any MAPI property using either a name or a MAPI property tag. For access with the property tag, use *objAddressEntry*.**Fields(***proptag***)**, where *proptag* is the 32-bit MAPI property tag associated with the property, such as **mapiPR_MESSAGE_CLASS**. To access a named property, use *objAddressEntry*.**Fields(***name***)**, where *name* is a string that represents the custom property name.

ID Property (AddressEntry Object)

The **ID** property returns the unique identifier of the AddressEntry object as a string. Read-only.

Syntax

objAddressEntry.**ID**

Data Type

String

Remarks

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

MAPI assigns a permanent, unique identifier when an object is created. This identifier does not change from one MAPI session to another, nor from one messaging domain to another.

The **ID** property corresponds to the MAPI property PR_ENTRYID, converted to a string of hexadecimal characters.

Example

This code fragment 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 Message     ' new msg; set its recipient using ID
Dim objNewRecip As Recipient ' new msg recipient; 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 Library"
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 ' make sure new data get saved in MAPI
objNewMsg.Send showDialog:=False
MsgBox "Created a new message in the Outbox and sent it"
Exit Function
' error handling
End Function
```

IsSameAs Method (AddressEntry Object)

The **IsSameAs** method returns **True** if the AddressEntry object is the same as the AddressEntry object being compared against.

Syntax

objAddressEntry.**IsSameAs(***objAddrEntry2***)**

Parameters

objAddressEntry
Required. This AddressEntry object.

objAddrEntry2
Required. The AddressEntry object being compared against.

Remarks

Two AddressEntry objects are considered to be the same if and only if they are instantiations of the same physical (persistent) object. Two objects with the same value are still considered different if they do not instantiate the same physical object, for example if one is a copy of the other. In such a case **IsSameAs** returns **False**.

The **IsSameAs** method ultimately calls one of the MAPI **CompareEntryIDs** methods to determine if two objects are the same. A generic comparison of any two object identifiers is also available with the Session object's **CompareIDs** method.

Members Property (AddressEntry Object)

The **Members** property returns an AddressEntries collection that contains the members of a distribution list. Read-only.

Syntax

objAddressEntry.**Members**

Data Type

Object

Remarks

The **Members** property returns a collection of all the members of the AddressEntry object if it is a distribution list. You can browse the returned AddressEntries collection, and you can add and delete entries if you have change access.

If the AddressEntry object is not a distribution list, the **Members** property returns **Nothing**.

Name Property (AddressEntry Object)

The **Name** property returns or sets the display name or alias of the AddressEntry object as a string. Read/write.

Syntax

objAddressEntry.**Name**

Data Type

String

Remarks

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 and **Resolve** method.

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

The **Name** property is the default property of an AddressEntry object, meaning that *objAddressEntry* is syntactically equivalent to *objAddressEntry*.**Name** in Visual Basic code.

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

Using Addresses

Type Property (AddressEntry Object)

The **Type** property specifies the address type, such as SMTP, fax, or X.400. Read/write.

Syntax

objAddressEntry.**Type**

Data Type

String

Remarks

The address type 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 can be combined to form the *full address*, the complete messaging address that appears in the Recipient object's **Address** property using the following syntax:

AddressType:*AddressValue*

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

Example

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

Update Method (AddressEntry Object)

The **Update** method saves changes to the AddressEntry object in the MAPI system.

Syntax

objAddressEntry.**Update([***makePermanent, refreshObject***])**

Parameters

objAddressEntry
Required. The AddressEntry object.

makePermanent
Optional. Boolean. A value of **True** indicates that the property cache is flushed and all changes are committed in the underlying address book. **False** indicates that the property cache is flushed but not committed to persistent storage. The default value is **True**.

refreshObject
Optional. Boolean. A value of **True** indicates that the property cache is reloaded from the values in the underlying address book. **False** indicates that the property cache is not reloaded. The default value is **False**.

Remarks

Changes to objects are not permanently saved in the MAPI system until you call the **Update** method with the *makePermanent* parameter set to **True**.

For improved performance, the OLE Messaging Library caches property changes in private storage and updates either the object or the underlying persistent storage only when you explicitly request such an update. For efficiency, you should make only one call to **Update** with its *makePermanent* parameter set to **True**.

The *makePermanent* and *refreshObject* parameters combine to cause the following changes:

	refreshObject = True	**refreshObject = False**
makePermanent = True	Commit all changes, flush the cache, and reload the cache from the address book.	Commit all changes and flush the cache (default combination).
makePermanent = False	Flush the cache and reload the cache from the address book.	Flush the cache.

Call **Update(False, True)** to flush the cache and then reload the values from the address book.

Example

The following code fragment changes the display name for a valid AddressEntry address:

```
' Function: AddressEntry_Update
' Purpose: Demonstrate the Update method
Function AddressEntry_Update()
Dim objRecipColl As Recipients ' Recipients collection
Dim objNewRecip As Recipient   ' 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)
' above could be objRecipColl(1) since .Item is default property
With objNewRecip.AddressEntry
    .Name = .Name & " the Magnificent"
    .Type = "X.500" ' you can also change the Type
    .Update         ' defaults to makePermanent = True
End With
MsgBox "Updated address entry name: " & objNewRecip.AddressEntry.Name
Exit Function
' error handling omitted
End Function
```

AddressEntryFilter Object

The AddressEntryFilter object specifies criteria for restricting a search on an AddressEntries collection.

Quick Info

Specified in header file:	MDISP.ODL
First available in:	OLE Messaging Library version 1.1
Parent objects:	AddressEntries collection
Child objects:	Fields collection
Default property:	Name

Properties

Name	Available in version	Type	Access
Address	1.1	String	Read/write
Application	1.1	String	Read-only
Class	1.1	Long	Read-only
Fields	1.1	Field object or Fields collection object	Read-only
Name	1.1	String	Read/write
Not	1.1	Boolean	Read/write
Or	1.1	Boolean	Read/write
Parent	1.1	AddressEntries collection object	Read-only
Session	1.1	Session object	Read-only

Methods

Name	Available in version	Parameters
IsSameAs	1.1	*object* as **Object**

Remarks

An AddressEntryFilter object with no criteria is created by default for every AddressEntries collection. This means that initially the filter's properties are unset and its child Fields collection is empty. You specify the filter by setting values for its properties, and by adding fields to its Fields collection and setting a value for each added field.

The filter is invoked when the AddressEntries collection is traversed with the **Get** methods. Each field participates in a MAPI search restriction comparing the field's **Value** property against the value of the AddressEntry property specified by the field's **ID** property.

For fields of data type other than **String**, the MAPI search restriction type is RES_PROPERTY with relational operator RELOP_EQ. For fields of data type **String**, the restriction type is RES_CONTENT with fuzzy level options FL_SUBSTRING, FL_IGNORECASE, and FL_LOOSE.

The results of the individual restrictions are normally **AND**ed together to form the final filter value. You can change this by setting the **Or** property, which causes all the results to be **OR**ed instead of **AND**ed. You can also set the **Not** property to specify that the result of each individual restriction is to be negated before being **AND**ed or **OR**ed into the final filter value.

The AddressEntryFilter object is persistent within its parent AddressEntries collection. It is not deleted even when it is released, and it remains attached to the AddressEntries collection until the collection's **Filter** property is set to **Nothing** or the collection is itself released.

Address Property (AddressEntryFilter Object)

The **Address** property specifies the *full address* for the AddressEntry object being filtered. Read/write.

Syntax

objAddressEntryFilter.**Address**

Data Type

String

Remarks

The AddressEntryFilter object's **Address** property is a concatenation of the address type and messaging address in the following format:

AddressType:*AddressValue*

where *AddressType* and *AddressValue* represent the AddressEntry object's **Type** and **Address** properties.

The AddressEntryFilter object's **Address** property corresponds to a combination of the MAPI properties PR_ADDRTYPE and PR_EMAIL_ADDRESS. It represents the *full address*, that is, the complete messaging address used by the MAPI system.

The **Address** property can be copied from the **Address** property of a Recipient. The advantage of doing this is that the value of the Recipient object's **Address** property has already been computed by the OLE Messaging Library from its **Name** property and the **Resolve** method.

Fields Property (AddressEntryFilter Object)

The **Fields** property returns a single Field object or a Fields collection object. Read-only.

Syntax

objAddressEntryFilter.**Fields**

objAddressEntryFilter.**Fields(***index***)**

objAddressEntryFilter.**Fields(***proptag***)**

objAddressEntryFilter.**Fields(***name***)**

index
Short integer (less than or equal to 65535). Specifies the index within the Fields collection.

proptag
Long integer (greater than or equal to 65536). Specifies the property tag value for the MAPI property to be retrieved.

name
String. Specifies the name of the custom MAPI property.

Data Type

Object

Remarks

The **Fields** property returns one or more of the fields associated with an AddressEntryFilter object. Each field typically corresponds to a MAPI property, and together the fields that have been added to the collection specify the filter.

The **Fields** property provides a generic access mechanism that allows Visual Basic and Visual C++ programmers to retrieve the value of a MAPI property using either its name or its MAPI property tag. For access with the property tag, use *objAddressEntryFilter*.**Fields(***proptag***)**, where *proptag* is the 32-bit MAPI property tag associated with the property, such as **mapiPR_MESSAGE_CLASS**. To access a named property, use *objAddressEntryFilter*.**Fields(***name***)**, where *name* is a string that represents the custom property name.

IsSameAs Method (AddressEntryFilter object)

The **IsSameAs** method returns **True** if the AddressEntryFilter object is the same as the AddressEntryFilter object being compared against.

Syntax

objAddressEntryFilter.**IsSameAs**(*objAddrEntryFilter2*)

Parameters

objAddressEntryFilter
Required. This AddressEntryFilter object.

objAddrEntryFilter2
Required. The AddressEntryFilter object being compared against.

Remarks

Two AddressEntryFilter objects are considered to be the same if and only if they are instantiations of the same physical (persistent) object. Two objects with the same value are still considered different if they do not instantiate the same physical object, for example if one is a copy of the other. In such a case **IsSameAs** returns **False**.

The **IsSameAs** method ultimately calls one of the MAPI **CompareEntryIDs** methods to determine if two objects are the same. A generic comparison of any two object identifiers is also available with the Session object's **CompareIDs** method.

Name Property (AddressEntryFilter Object)

The **Name** property specifies a value for use in an ANR (ambiguous name resolution) restriction on an AddressEntry object. Read/write.

Syntax

objAddressEntryFilter.**Name**

Data Type

String

Remarks

The **Name** property contains an ambiguous name resolution (ANR) string that can be compared against each AddressEntry object using a provider-defined algorithm. The property or properties used in the comparison are chosen by the provider as part of the algorithm; the PR_DISPLAY_NAME property is the most commonly used.

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

The **Name** property is the default property of an AddressEntryFilter object, meaning that *objAddressEntryFilter* is syntactically equivalent to *objAddressEntryFilter*.**Name** in Visual Basic code.

Not Property (AddressEntryFilter Object)

The **Not** property specifies that all restriction values are to be negated before being **AND**ed or **OR**ed to specify the address entry filter. Read/write.

Syntax

objAddressEntryFilter.**Not**

Data Type

Boolean

Remarks

If the **Not** property is **False**, the restriction values are treated normally. If it is **True**, each value is toggled (between **True** and **False**) before being used.

Or Property (AddressEntryFilter Object)

The **Or** property specifies that the restriction values are to be **OR**ed instead of **AND**ed to specify the address entry filter. Read/write.

Syntax

objAddressEntryFilter.**Or**

Data Type

Boolean

Remarks

If the **Or** property is **False**, all the restriction values are **AND**ed together. If it is **True**, the values are **OR**ed together.

AddressList Object

The AddressList object supplies a list of address entries to which a messaging system can deliver messages.

Quick Info

Specified in header file:	MDISP.ODL
First available in:	OLE Messaging Library version 1.1
Parent objects:	AddressLists collection
Child objects:	AddressEntries collection
Default property:	Name

Properties

Name	Available in version	Type	Access
AddressEntries	1.1	AddressEntries collection object	Read-only
Application	1.1	String	Read-only
Class	1.1	Long	Read-only
ID	1.1	String	Read-only
Index	1.1	Long	Read-only
IsReadOnly	1.1	Boolean	Read-only
Name	1.1	String	Read-only
Parent	1.1	AddressLists collection object	Read-only
Session	1.1	Session object	Read-only

Methods

Name	Available in version	Parameters
IsSameAs	1.1	*object* as **Object**

Remarks

An AddressList object represents one address book container available under the MAPI address book hierarchy for the current session. The entire hierarchy is available through the parent AddressLists collection.

AddressEntries Property (AddressList Object)

The **AddressEntries** property returns a single AddressEntry object or an AddressEntries collection object. Read-only.

Syntax

Set *objAddrEntriesColl* **=** *objAddressList.***AddressEntries**

Set *objOneAddrEntry* = *objAddressList.***AddressEntries(***index***)**

objAddrEntriesColl
Object. An AddressEntries collection object.

objAddressList
Object. The AddressList object.

objOneAddrEntry
Object. A single AddressEntry object.

index
Long. Specifies the number of the address entry within the AddressEntries collection. Ranges from 1 to the size of the collection.

Data Type

Object

Remarks

An AddressEntries collection is a large collection, and its size cannot necessarily be determined from its **Count** property. It is not safe to use the *index* parameter with the **AddressEntries** property unless an indexed loop has determined that an address entry at that position in the collection actually exists.

ID Property (AddressList Object)

The **ID** property returns the unique identifier of the AddressList object as a string. Read-only.

Syntax

objAddressList.**ID**

Data Type

String

Remarks

MAPI assigns a permanent, unique identifier when an object is created. This identifier does not change from one MAPI session to another, nor from one messaging domain to another.

The **ID** property corresponds to the MAPI property PR_ENTRYID, converted to a string of hexadecimal characters.

Example

This code fragment displays the value of the AddressList object's permanent identifier:

```
Dim strAddressListID as String ' hex string version of ID
Dim objAddressList as AddressList ' assume valid for this example
strAddressListID = objAddressList.ID ' global variable
MsgBox "Address Book ID = " & strAddressListID
```

Index Property (AddressList Object)

The **Index** property returns the index number for the AddressList object within the AddressLists collection. Read-only.

Syntax

objAddressList.**Index**

Data Type

Long

Remarks

The **Index** property indicates this object's position within the parent AddressLists Collection.

Example

```
Function AddressListsGetByIndex()
Dim rqIndex As Long ' requested index value within collection
Dim svIndex As Long ' saved index value within collection
Dim objOneAddressList As AddressList
    ' set error handler here
    If objAddressListsColl Is Nothing Then
        MsgBox "must select a valid AddressLists collection"
        Exit Function
    End If
    If 0 = objAddressListsColl.Count Then
        MsgBox "must select collection with 1 or more address lists"
        Exit Function
    End If
    ' prompt user for rqIndex
    Set objOneAddressList = objAddressListsColl.Item(rqIndex)
    MsgBox "Selected address list: " & objOneAddressList.Name
    svIndex = objOneAddressList.Index ' save index for later
    ' get same AddressList object later
    Set objOneAddressList = objAddressListsColl.Item(svIndex)
    If objOneAddressList Is Nothing Then
        MsgBox "Error: could not reselect the address list"
    Else
        MsgBox "Reselected address list (" & svIndex & _
            ") using saved index: " & objOneAddressList.Name
    End If
    Exit Function
```

IsReadOnly Property (AddressList Object)

The **IsReadOnly** property indicates that the AddressList object cannot be modified. Read-only.

Syntax

objAddressList.**IsReadOnly**

Data Type

Boolean

Remarks

The **IsReadOnly** property refers to adding and deleting the entries in the address book container represented by the AddressList object. The property is **True** if no entries can be added or deleted, and **False** if the container can be modified, that is, if address entries can be added to and deleted from the container.

IsReadOnly refers to the address book entries in the context of the address book container. It does not indicate whether the contents of the individual entries themselves can be modified.

IsSameAs Method (AddressList Object)

The **IsSameAs** method returns **True** if the AddressList object is the same as the AddressList object being compared against.

Syntax

objAddressList.**IsSameAs(***objAddrList2***)**

Parameters

objAddressList
Required. This AddressList object.

objAddrList2
Required. The AddressList object being compared against.

Remarks

Two AddressList objects are considered to be the same if and only if they are instantiations of the same physical (persistent) object. Two objects with the same value are still considered different if they do not instantiate the same physical object, for example if one is a copy of the other. In such a case **IsSameAs** returns **False**.

The **IsSameAs** method ultimately calls one of the MAPI **CompareEntryIDs** methods to determine if two objects are the same. A generic comparison of any two object identifiers is also available with the Session object's **CompareIDs** method.

Name Property (AddressList Object)

The **Name** property returns the name of the AddressList object as a string. Read-only.

Syntax

objAddressList.**Name**

Data Type

String

Remarks

The **Name** property corresponds to the MAPI property PR_DISPLAY_NAME for the address book container represented by the AddressList object.

The **Name** property is the default property of an AddressList object, meaning that *objAddressList* is syntactically equivalent to *objAddressList*.**Name** in Visual Basic code.

Example

```
Dim objAddressList As Object ' assume valid address list object
MsgBox "Address book container name = " & objAddressList.Name
```

AddressLists Collection Object

The AddressLists collection object contains one or more AddressList objects.

Quick Info

Specified in header file:	MDISP.ODL
First available in:	OLE Messaging Library version 1.1
Parent objects:	Session
Child objects:	AddressList
Default property:	Item

An AddressLists collection is considered a *small collection*, which means that it supports count and index values that let you access an individual AddressList object through the **Item** property. The AddressLists collection supports the Visual Basic **For Each** statement.

Properties

Name	Available in version	Type	Access
Application	1.1	String	Read-only
Class	1.1	Long	Read-only
Count	1.1	Long	Read-only
Item	1.1	AddressList object	Read-only
Parent	1.1	Session object	Read-only
Session	1.1	Session object	Read-only

Methods

(None.)

Remarks

The AddressLists collection provides access to the root of the MAPI address book hierarchy for the current session. You can obtain the collection through the parent Session object's **AddressLists** property.

You can use the **Count** and **Item** properties to traverse the hierarchy for all available address books, or you can use the **Item** property to select a particular AddressList object. The type of access you obtain depends on the access granted to you by each available address book provider.

Each AddressList object represents one MAPI address book container. The AddressLists collection contains only those AddressList objects that contain recipients, and not those containing only subcontainers. For more information on the different types of containers, see the description of the PR_CONTAINER_FLAGS property in the *MAPI Programmer's Reference*.

Count Property (AddressLists Collection)

The **Count** property returns the number of AddressList objects in the collection. Read-only.

Syntax

objAddrListsColl.**Count**

Data Type

Long

Example

This code fragment uses the **Count** and **Item** properties to determine how many AddressList objects are available in the collection:

```
Dim i As Integer ' loop counter
Set hierarchy = MAPI.Session.AddressLists
' make sure returned collection object is valid
If hierarchy Is Nothing Then
    ' Exit "Address book hierarchy is invalid"
End If
' see if hierarchy is empty
i = hierarchy.Count ' count of address books in hierarchy
If 0 = i Then
    ' Exit "No available address books"
End If
' precautionary loop to make sure address books are all valid
For i = 1 To hierarchy.Count Step 1
    If Nothing = hierarchy.Item(i) Then
    ' Exit "Address book is invalid"
    End If
Next i
```

Item Property (AddressLists Collection)

The **Item** property returns a single AddressList object from the AddressLists collection. Read-only.

Syntax

objAddrListsColl.**Item(***index***)**

objAddrListsColl.**Item(***name***)**

index
A long integer ranging from 1 to *objAddrListsColl*.**Count**.

name
A string representing the value of the **Name** property of an AddressList object.

Data Type

Object

Remarks

The **Item** property works like an accessor property for small collections.

The **Item(***index***)** syntax selects an arbitrary AddressList object within the AddressLists collection. The example in the **Count** property shows how these two properties can be used together to traverse the collection.

The **Item(***name***)** syntax returns the first AddressList object whose **Name** property matches the string specified by *name*.

The **Item** property is the default property of an AddressLists collection, meaning that *objAddrListsColl***(***index***)** is syntactically equivalent to *objAddrListsColl*.**Item(***index***)** in Visual Basic code.

Attachment Object

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

Quick Info

Specified in header file:	MDISP.ODL
First available in:	OLE Messaging Library version 1.0.a
Parent objects:	Attachments collection
Child objects:	(none)
Default property:	Name

Properties

Name	Available in version	Type	Access
Application	1.0.a	String	Read-only
Class	1.0.a	Long	Read-only
Fields	1.1	Field object or Fields collection object	Read-only
Index	1.0.a	Long	Read-only
Name	1.0.a	String	Read/write
Parent	1.0.a	Attachments collection object	Read-only
Position	1.0.a	Long	Read/write
Session	1.0.a	Session object	Read-only
Source	1.0.a	String or Message object	Read/write
Type	1.0.a	Long	Read/write

Methods

Name	Available in version	Parameters
Delete	1.0.a	(none)
IsSameAs	1.1	*object* as **Object**
ReadFromFile	1.0.a	*fileName* as **String**
WriteToFile	1.0.a	*fileName* as **String**

Delete Method (Attachment Object)

The **Delete** method deletes the Attachment object.

Syntax

objAttachment.**Delete()**

Parameters

objAttachment
Required. The Attachment object.

Remarks

The Attachment object is invalidated in memory, but the change is not permanent until you use the **Update**, **Send**, or **Delete** method on the Message object to which this attachment belongs.

The **Delete** operation invalidates the Attachment object but does not remove it from memory. The programmer should set the invalidated object to **Nothing** to remove it from memory, or reassign it to another attachment. Attempted access to a deleted object results in a return of **mapiE_INVALID_OBJECT**.

The immediate parent of this Attachment object is an Attachments collection, which is a child of the message. You can delete all the message's attachments by calling the collection's **Delete** method.

Fields Property (Attachment Object)

The **Fields** property returns a single Field object or a Fields collection object. Read-only.

Syntax

objAttachment.**Fields**

objAttachment.**Fields(***index***)**

objAttachment.**Fields(***proptag***)**

objAttachment.**Fields(***name***)**

index
Short integer (less than or equal to 65535). Specifies the index within the Fields collection.

proptag
Long integer (greater than or equal to 65536). Specifies the property tag value for the MAPI property to be retrieved.

name
String. Specifies the name of the custom MAPI property.

Data Type

Object

Remarks

The **Fields** property returns one or more of the fields associated with an Attachment object. Each field typically corresponds to a MAPI property.

The **Fields** property provides a generic access mechanism that allows Visual Basic and Visual C++ programmers to retrieve the value of a MAPI property using either its name or its MAPI property tag. For access with the property tag, use *objAttachment*.**Fields(***proptag***)**, where *proptag* is the 32-bit MAPI property tag associated with the property, such as **mapiPR_MESSAGE_CLASS**. To access a named property, use *objAttachment*.**Fields(***name***)**, where *name* is a string that represents the custom property name.

Index Property (Attachment Object)

The **Index** property returns the index number for the Attachment object within the Attachments collection. Read-only.

Syntax

objAttachment.**Index**

Data Type

Long

Remarks

The **Index** property indicates this attachment's position within the parent Attachments collection. It can later be used to reselect this attachment with the collection's **Item** property.

An index value should not be considered a static value that remains constant for the duration of a session. It can be affected when other attachments are added and deleted. The index value is changed following an update to the Message object to which the Attachments collection belongs.

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
```

IsSameAs Method (Attachment Object)

The **IsSameAs** method returns **True** if the Attachment object is the same as the Attachment object being compared against.

Syntax

objAttachment.**IsSameAs(***objAttach2***)**

Parameters

objAttachment
Required. This Attachment object.

objAttach2
Required. The Attachment object being compared against.

Remarks

Two Attachment objects are considered to be the same if and only if they are instantiations of the same physical (persistent) object. Two objects with the same value are still considered different if they do not instantiate the same physical object, for example if one is a copy of the other. In such a case **IsSameAs** returns **False**.

The **IsSameAs** method ultimately calls one of the MAPI **CompareEntryIDs** methods to determine if two objects are the same. A generic comparison of any two object identifiers is also available with the Session object's **CompareIDs** method.

Name Property (Attachment Object)

The **Name** property returns or sets the display name of the Attachment object as a string. Read/write.

Syntax

objAttachment.**Name**

Data Type

String

Remarks

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

The **Name** property is the default property of an Attachment object, meaning that *objAttachment* is syntactically equivalent to *objAttachment*.**Name** in Visual Basic code.

Example

See the example for the Attachment object's **Index** property.

Position Property (Attachment Object)

The **Position** property returns or sets the position of the attachment within the text of the message. Read/write.

Syntax

objAttachment.**Position**

Data Type

Long

Remarks

The **Position** property is a long integer describing where the attachment should be placed in the message text. The attachment overwrites the character present at that position. Applications cannot place two attachments in the same location within a message, and attachments cannot be placed beyond the end of the message text.

The OLE Messaging Library does not manage rendering of the attachment within the message. The **Position** property simply provides directions for the rendering application.

The value -1 indicates that the attachment is not rendered using the **Position** property. The value 0, and all positive values, indicate an index to the text character within the message.

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 beginning 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)

The **ReadFromFile** method loads the contents of an attachment from a file.

Syntax

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

Parameters

objAttachment
Required. The Attachment object.

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

Remarks

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

The **ReadFromFile** method operates differently, depending on the value of the Attachment object's **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.
mapiEmbeddedMessage	(Not supported)

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

Note OLE Messaging Library versions 1.0.a and 1.1 do not support **ReadFromFile** for **mapiFileLink** or **mapiEmbeddedMessage** attachments. These calls generate the run-time error **mapiE_NO_SUPPORT**.

You can load the contents of an attachment when you first create it by specifying the *type* and *source* parameters when you call the **Add** of the Attachments collection.

Source Property (Attachment Object)

The **Source** property returns or sets the full path and file name, OLE class name, or unique message identifier for the attachment. Read/write.

Syntax

objAttachment.**Source**

Data Type

String

Remarks

The **Source** property returns or sets the full path and file name of the attachment data file for **mapiFileData** and **mapiFileLink** attachments. It returns or sets the OLE class name of the attachment for **mapiOLE** attachments. For **mapiEmbeddedMessage** attachments, the **Source** property is set with the **ID** property of the message to be embedded, and it returns the Message object itself. An embedded message is copied into the attachment at creation time.

The OLE Messaging Library does not synchronize the **Source** property and the **ReadFromFile** method. For **mapiFileData** and **mapiOLE** attachments, when you change the **Source** property to indicate a different file, you must also explicitly call the **ReadFromFile** method to update the object data. Similarly, when you call **ReadFromFile** with data from a different file, you must change the **Source** property.

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

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.”
mapiEmbeddedMessage	Specifies the unique identifier of the embedded message; returns the embedded Message object.

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

The **Source** property can be set at the time of creation of the attachment by supplying the *source* parameter to the **Add** method of the Attachments collection.

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 beginning 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

WriteToFile Method (Attachment Object)

Type Property (Attachment Object)

The **Type** property describes the attachment type. Read/write.

Syntax

objAttachment.**Type**

Data Type

Long

Remarks

The following 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.
mapiEmbeddedMessage	4	Attachment is an embedded message.

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

The **Type** property can be set at the time of creation of the attachment by supplying the *type* parameter to the **Add** method of the Attachments collection.

The **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 beginning 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

ReadFromFile Method (Attachment Object), **WriteToFile** Method (Attachment Object)

WriteToFile Method (Attachment Object)

The **WriteToFile** method 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.

Remarks

The **WriteToFile** method overwrites the file without warning if a file of that name already exists. Your application should check for the existence of the file before calling **WriteToFile**.

The **WriteToFile** method operates differently, depending on the value of the Attachment object's **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.
mapiEmbeddedMessage	(Not supported)

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

Note OLE Messaging Library version 1.1 does not support **WriteToFile** for **mapiFileLink** or **mapiEmbeddedMessage** attachments. These calls generate the run-time error **mapiE_NO_SUPPORT**.

See Also

ReadFromFile Method (Attachment Object)

Attachments Collection Object

The Attachments collection object contains one or more Attachment objects.

Quick Info

Specified in header file:	MDISP.ODL
First available in:	OLE Messaging Library version 1.0.a
Parent objects:	Message
Child objects:	Attachment
Default property:	Item

An Attachments collection is considered a *small collection*, which means that it supports count and index values that let you access an individual Attachment objects through the **Item** property. The Attachments collection supports the Visual Basic **For Each** statement.

Properties

Name	Available in version	Type	Access
Application	1.0.a	String	Read-only
Class	1.0.a	Long	Read-only
Count	1.0.a	Long	Read-only
Item	1.0.a	Attachment object	Read-only
Parent	1.0.a	Message object	Read-only
Session	1.0.a	Session object	Read-only

Methods

Name	Available in version	Parameters
Add	1.0.a	(optional) *name* as **String**, (optional) *position* as **Long**, (optional) *type* as **Long**, (optional) *source* as **String**
Delete	1.0.a	(none)

Add Method (Attachments Collection)

The **Add** method creates and returns 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. The default value is an empty string. To allow a user to click on the attachment that appears in the message and activate an associated application, supply the full file name, including the file extension.

position
Optional. Long. The position of the attachment within the body text of the message. The default value is zero.

type
Optional. Long. The type of attachment; either **mapiFileData**, **mapiFileLink**, **mapiOLE**, or **mapiEmbeddedMessage**. The default value is **mapiFileData**.

source
Optional. String. The path and file name of the file containing the data for the attachment, or the unique identifier of the message to be embedded. The path and file name must be in the appropriate format for the attachment type, specified by the *type* parameter. The default value is an empty string.

Remarks

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. The data is read into the attachment.
mapiFileLink	Specifies a full path and file name in a universal naming convention (UNC) format, such as \ \SALES\INFO\PRODUCTS\NEWS.DOC. The attachment is a link, so the **Add** method does not read the data.
mapiOLE	Specifies a full path and file name to a valid OLE docfile, for example C: \DOCUMENT\BUDGET2.XLS. The data is read into the attachment.
mapiEmbeddedMessage	Specifies the **ID** property of the message to

be embedded. The message is copied into the attachment.

When the *type* parameter has the value **mapiFileLink**, the *source* parameter is a full path and file name in a UNC format. This is suitable for sending attachments to recipients who have access to a common file server. Note that when you use the *type* **mapiFileLink**, the OLE Messaging Library does not validate the file name.

If you do not specify the *type* and *source* parameters when you call the **Add** method, you must later explicitly set these properties. For **mapiFileData** and **mapiOLE** types, you must also call the **ReadFromFile** method on the new Attachment object to load the attachment's content.

The **Index** property of the new Attachment object equals the new **Count** property of the Attachments collection.

The attachment is saved in the MAPI system when you **Update** or **Send** the parent Message object.

Count Property (Attachments Collection)

The **Count** property returns the number of Attachment objects in the collection. Read-only.

Syntax

objAttachColl.**Count**

Data Type

Long

Example

This code fragment stores in an array the names of all Attachment objects in the collection. It shows the **Count** and **Item** properties working together.

```
' 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
```

Delete Method (Attachments Collection)

The **Delete** method deletes all the attachments in the Attachments collection.

Syntax

objAttachColl.**Delete()**

Parameters

objAttachColl
Required. The Attachments collection object.

Remarks

The **Delete** operation invalidates all the Attachment objects in the collection but does not remove them from memory. The programmer should set the invalidated objects to **Nothing** to remove them from memory, or reassign them to other attachments. Attempted access to a deleted object results in a return of **mapiE_INVALID_OBJECT**.

Be cautious using the **Delete** method with a collection, because it deletes all the collection's member objects. To delete only one Attachment object, use the **Delete** method specific to that object.

The effect of the **Delete** method is not permanent until you use the **Update**, **Send**, or **Delete** method on the parent Message object containing the Attachments collection. Any member once permanently deleted cannot be recovered. However, the collection itself it still valid, and you can **Add** new members to it.

Item Property (Attachments Collection)

The **Item** property returns a single Attachment object from the Attachments collection. Read-only.

Syntax

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

index

A long integer that ranges from 1 to *objAttachColl*.**Count**, or a string that specifies the name of the object.

Data Type

Object

Remarks

The **Item** property works like an accessor property for small collections.

The **Item** property is the default property of an Attachments collection, meaning that *objAttachColl***(***index***)** is syntactically equivalent to *objAttachColl*.**Item(***index***)** in Visual Basic code.

Example

This code fragment shows the **Count** and **Item** properties working together:

```
' 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
Dim i As Integer ' loop counter
    ' 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
```

Field Object

A Field object represents a MAPI property on an OLE Messaging Library object.

Quick Info

Specified in header file:	MDISP.ODL
First available in:	OLE Messaging Library version 1.0.a
Parent objects:	Fields collection
Child objects:	(none)
Default property:	Value

Properties

Name	Available in version	Type	Access
Application	1.0.a	String	Read-only
Class	1.0.a	Long	Read-only
ID	1.0.a	Long	Read-only
Index	1.0.a	Long	Read-only
Name	1.0.a	String	Read-only
Parent	1.0.a	Fields collection object	Read-only
Session	1.0.a	Session object	Read-only
Type	1.0.a	Integer	Read-only
Value	1.0.a	Variant	Read/write

Methods

Name	Available in version	Parameters
Delete	1.0.a	(none)
ReadFromFile	1.0.a	*fileName* as **String**
WriteToFile	1.0.a	*fileName* as **String**

Remarks

The Field object gives you the ability to add or access MAPI properties of an AddressEntry, AddressEntryFilter, Folder, Message, or MessageFilter object.

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

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

The Field object also supports multivalued MAPI properties. The multivalued property appears to the

Visual Basic application as a variant array; that is, you can use the **For ... Next** statement to access individual array entries, as shown in the following sample program.

```
Dim rgstr(0 To 9) As String
' Build array of values for MV prop
For i = 0 To 9
    rgstr(i) = "String" + Str(i)
Next

' Create MV field on the message (note that we don't specify
' the array as third argument to Fields.Add, but add separately)
Set f = msg.Fields.Add("FancyName", vbString + vbArray)
f.Value = rgstr ' Set value of the new field
' Save/send the message, logoff, etc.

' later: code that reads the multivalued properties
Dim rgstr As Variant
Set f = msg.Fields.Item("FancyName") ' Get MV Field
rgstr = f.Value ' Get array of values into a variant
For i = LBound(rgret) To UBound(rgret)
    MsgBox rgret(i)
Next I
```

For more information on MAPI properties, see the reference documentation for the Fields collection and the *MAPI Programmer's Reference*.

Delete Method (Field Object)

The **Delete** method deletes the user-defined or optional Field object.

Syntax

objField.**Delete**

Parameters

objField
Required. The Field object.

Remarks

This method only deletes user-defined fields and fields that represent properties considered optional by the underlying provider.

The field is invalidated in memory, but the change is not permanent until you use the **Update**, **Send**, or **Delete** method on the parent AddressEntry, AddressEntryFilter, Folder, Message, or MessageFilter object of the Fields collection.

The **Delete** operation invalidates the Field object but does not remove it from memory. The programmer should set the invalidated object to **Nothing** to remove it from memory, or reassign it to another field. Attempted access to a deleted object results in a return of **mapiE_INVALID_OBJECT**.

See Also

Add Method (Fields Collection)

ID Property (Field Object)

The **ID** property returns the MAPI tag for the Field object as a long integer. Read-only.

Syntax

objField.**ID**

Data Type

Long

Remarks

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

A MAPI property tag is a 32-bit unsigned integer. Its high-order 16 bits contain the MAPI property identifier, and its low-order 16 bits contain the MAPI property type. For more information, see “About Property Tags” in the *MAPI Programmer’s Reference*.

Note The MAPI property type is not the same as the OLE Messaging Library **Type** property. There is a correspondence between the two entities, but their value sets are not the same. The Field object **ID** property contains the MAPI property type; its **Type** property contains the OLE Messaging Library Visual Basic data type.

Example

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

Index Property (Field Object)

The **Index** property returns the index number of the Field object within the Fields collection. Read-only.

Syntax

objField.**Index**

Data Type

Long

Remarks

An index value should not be considered a static value that remains constant for the duration of a session. It can be affected when other fields are added and deleted. The index value is changed following an update to the object to which the Fields collection belongs.

Example

This code fragment shows the Fields collection's **Count** property and the **Index** property working together:

```
' 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, then loop back ...
```

Name Property (Field Object)

The **Name** property returns the name of the field as a string. Read-only.

Syntax

objField.**Name**

Data Type

String

Remarks

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's **Add** method.

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
' later: fragment from Field_Name; modified to use objNewField
    If "" = objNewField.Name Then
        MsgBox "Field has no name; ID = " & objNewField.ID
    Else
        MsgBox "Field name = " & objNewField.Name
    End If
```

ReadFromFile Method (Field Object)

The **ReadFromFile** method loads the value of a string or binary field from a file.

Syntax

objField.**ReadFromFile(***fileName***)**

Parameters

objField
Required. The Field object.

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

Remarks

The **ReadFromFile** method reads the string or binary value from the specified file 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 **vbInteger**, **vbLong**, and **vbBoolean**. 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 object is not **vbString** or **vbBlob**.

Note that some binary types are converted to a hexadecimal 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 when the values are equivalent.

In addition, support for types can vary among providers. Not all providers support both the **vbString** and **vbBlob** property types.

ReadFromFile returns **mapiE_INTERFACE_NOT_SUPPORTED** for Field objects obtained from a Folder object’s Fields collection.

See Also

WriteToFile Method (Field Object)

Type Property (Field Object)

The **Type** property returns or sets the data type of the Field object. Read/write.

Syntax

objField.**Type**

Data Type

Integer

Remarks

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

Valid data types are as follows:

Type	Description	Decimal 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, PT_I8
vbCurrency	Scaled integer (8 bytes)	6	VT_CY	PT_CURRENCY
vbDate	Date/time (8 bytes)	7	VT_DATE	PT_APPTIME, PT_SYSTIME
vbString	String	8	VT_BSTR	PT_TSTRING, PT_BINARY
vbBoolean	Boolean	11	VT_BOOL	PT_BOOLEAN
vbDataObject	Data object	13	VT_UNKNOWN	PT_OBJECT
vbBlob	Binary (unknown format)	65	VT_BLOB	PT_BINARY
vbArray	Multivalued type	8192	VT_ARRAY	PT_MV_(type)

Note that the types **vbNull** and **vbDataObject** are not supported in version 1.0.a.

The **vbArray** data type must always be used in conjunction with one of the other types, for example **vbArray + vbInteger**. When you use a multivalued type, to avoid a **mapiE_INVALID_TYPE** error you

must also declare the array to be of the appropriate type:

```
Dim Words(10) As String ' NOT just Dim Words(10)
' ...
Set objKeysField = objFieldsColl.Add("Keywords", vbArray + vbString)
objKeysField.Value = Words
```

Note that MAPI stores all custom properties that represent date and time information using Greenwich Mean Time (GMT). The OLE Messaging Library converts these properties so that the values appear to the user in local time.

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

Value Property (Field Object)

The **Value** property returns or sets the value of the Field object. Read/write.

Syntax

objField.**Value**

Data Type

Variant

Remarks

The **Value** property 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 values **True** or **False**. When the Field object has the **Type** property **vbInteger**, the **Value** property can contain a short integer.

The **Value** property is the default property of a Field object, meaning that *objField* is syntactically equivalent to *objField*.**Value** in Visual Basic code.

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
```

WriteToFile Method (Field Object)

The **WriteToFile** method saves the field value to a file in the file system.

Syntax

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

Parameters

objField

Required. The Field object.

fileName

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

Remarks

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

Note that **WriteToFile** is not supported for simple types, such as **vbInteger**, **vbLong**, and **vbBoolean**. 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 object is not **vbString** or **vbBlob**.

Note that some binary types are represented in hexadecimal string format by the OLE Messaging Library but written to persistent storage in binary format. Comparison operations on the **Value** property and the actual contents of the file can return “not equal” even when the values are equivalent.

In addition, support for types can vary among providers. Not all providers support both the **vbString** and **vbBlob** property types.

See Also

ReadFromFile Method (Field Object)

Fields Collection Object

The Fields collection object contains one or more Field objects.

Quick Info

Specified in header file:	MDISP.ODL
First available in:	OLE Messaging Library version 1.0.a
Parent objects:	AddressEntry AddressEntryFilter Folder (Inbox or Outbox) Message MessageFilter
Child objects:	Field
Default property:	Item

A Fields collection is considered a *small collection*, which means that it supports count and index values that let you access an individual Field object through the **Item** property. The Fields collection supports the Visual Basic **For Each** statement.

Properties

Name	Available in version	Type	Access
Application	1.0.a	String	Read-only
Class	1.0.a	Long	Read-only
Count	1.0.a	Long	Read-only
Item	1.0.a	Field object	Read-only
Parent	1.0.a	AddressEntry object, Inbox or Outbox folder object, or Message object	Read-only
Session	1.0.a	Session object	Read-only

Methods

Name	Available in version	Parameters
Add	1.0.a	*name* as **String**, *Class* as **Long**, *PropTag* as **Long**, (optional) *value* as **Variant**, (optional) *PropsetID* as **String**
Delete	1.0.a	(none)
SetNamespace	1.0.a	(optional) *PropsetID* as **String**

Remarks

Field objects give you the ability to access MAPI properties on the parent object of the Fields collection. These include the predefined underlying MAPI properties and your own custom user-defined properties.

MAPI defines a set of properties with identifiers less than the value 0x8000. These are known as *unnamed properties* because they are usually accessed using the MAPI property tag rather than a

name. You can access these MAPI-defined properties using the Fields collection. All MAPI properties are accessible except those of types PT_OBJECT and PT_CLSID.

You can also extend the properties available through MAPI by defining your own properties. These user-defined properties, defined using a name and automatically assigned an identifier greater than 0x8000 by the OLE Messaging Library, are known as *named properties*. (C++ programmers can access the property name in the MAPI structure **MAPINAMEID** and convert it to the property tag value.)

All named properties are defined as part of a *property set*, which is also known in the context of the OLE Messaging Library as a *namespace*.

A property set is defined by a **GUID**, or globally unique identifier. The OLE Messaging Library represents this **GUID** as a string of hexadecimal characters. Such identifiers are usually referenced using a constant that starts with the characters PS_, such as PS_PUBLIC_STRINGS, the default property set for all properties created using the OLE Messaging Library.

You can also choose to organize your custom properties within their own semantic space by defining your own property set. The **Add** and **SetNamespace** methods and the **Item** property let you specify the property set identifier to be used for named property access.

When creating your own property set, you should be aware that MAPI reserves several property set identifiers for specific purposes. The following table lists reserved property sets:

Reserved Property Set	Description
PS_MAPI	Allows providers to supply names for the unnamed properties (properties with identifiers less than 0x8000).
PS_PUBLIC_STRINGS	Default property set for custom properties added using the OLE Messaging Library.
PS_ROUTING_ADDRTYPE	E-mail address types that are translated between messaging domains.
PS_ROUTING_DISPLAY_NAME	Display name properties that are translated between messaging domains.
PS_ROUTING_EMAIL_ADDRESSES	E-mail addresses that are translated between messaging domains.
PS_ROUTING_ENTRYID	Long-term entry identifiers that are translated between messaging domains.
PS_ROUTING_SEARCH_KEY	Search keys that are translated between messaging domains.

To create your own **GUID** that identifies your property set, you can either use the Win32 command-line utility **UUIDGEN** or you can call the OLE function **CoCreateGuid** to supply one for you, as demonstrated in the following code fragment:

```
' declarations required for the call to CoCreateGuid
Type GUID
    Guid1 As Long
    Guid2 As Long
    Guid3 As Long
    Guid4 As Long
End Type
Declare Function CoCreateGuid Lib "OLE32.DLL" (pGuid As GUID) As Long
```

```
Global Const S_OK = 0
Dim strPropID As String
Dim lResult As Long
Dim lGuid As GUID

' call CoCreateGuid, then convert the result to a hex string
    lResult = CoCreateGuid(lGuid)
    If lResult = S_OK Then
        strPropID =  Hex$(lGuid.Guid1) & Hex$(lGuid.Guid2)
        strPropID = myHexString & Hex$(lGuid.Guid3)
        strPropID = myHexString & Hex$(lGuid.Guid4)
    Else
        ' ... handle error ...
    End If
```

For more information on named properties and property sets, see “Named Properties” in the *MAPI Programmer’s Reference*. For more information on **UUIDGEN** and **CoCreateGuid**, see the Win32 SDK documentation.

Note that MAPI stores all custom properties that represent date and time information using Greenwich Mean Time (GMT). The OLE Messaging Library converts these properties so that the values appear to the user in local time.

Example

To uniquely identify a Field object in the Fields collection, use the Field object’s **Name** or **Index** property:

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

See Also

Object Collections

Add Method (Fields Collection)

The **Add** method creates and returns a new Field object in the Fields collection.

Syntax

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

Set *objfield* = *objFieldsColl*.**Add (***PropTag* **[***, value***])**

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's **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	Scaled integer (8 bytes)	6	VT_CY
vbDate	Date/time (8 bytes)	7	VT_DATE
vbString	String	8	VT_BSTR
vbBoolean	Boolean	11	VT_BOOL
vbDataObject	Data object	13	VT_UNKNOWN
vbBlob	Blob (unknown format)	65	VT_BLOB
vbArray	Multivalued type	8192	VT_ARRAY

PropTag

Required. Long. The MAPI property tag for the corresponding MAPI property.

value

Optional. Variant. The value of the field, of the data type specified in the *Class* parameter. When no value is supplied, no data is present for the object. You must make subsequent calls to the Field object's **ReadFromFile** method.

PropsetID

Optional. String. Specifies the identifier of the property set, represented as a string of hexadecimal characters. When the identifier is not present, the property is created within the default property set. The default property set is either the property set specified to the **SetNamespace** method, or the initial default property set value, PS_PUBLIC_STRINGS.

Remarks

Support for the **Add** method is provider-dependent. The second syntax should be used when adding a MAPI property to the Fields collection.

The *name*, *Class*, and *value* parameters in the first syntax correspond to the **Name**, **Type**, and **Value** properties of the Field object.

The *PropTag* parameter in the second syntax contains the 32-bit MAPI property tag associated with the property and corresponds to the **ID** property of the Field object. The property tag contains the MAPI property identifier in its high-order 16 bits and the MAPI property type in its low-order 16 bits. All MAPI properties are accessible except those of MAPI type PT_OBJECT or PT_CLSID.

The **Index** property of the new Field object equals the new **Count** property of the Fields collection.

The field is saved in the MAPI system when you **Update** the parent object, or **Send** it if the Fields collection's parent is a Message object.

The **vbArray** data type must always be used in conjunction with one of the other types, for example **vbArray + vbInteger**. When you use a multivalued type, to avoid a **mapiE_INVALID_TYPE** error you must also declare the array to be of the appropriate type:

```
Dim Words(10) As String ' NOT just Dim Words(10)
' ...
Set objKeysField = objFieldsColl.Add("Keywords", vbArray + vbString)
objKeysField.Value = Words
```

When you use the **vbBlob** type for binary data, you supply the value in the form of a hexadecimal string that contains the hexadecimal representation of the bytes in the binary object (such as a hexadecimal dump of the object).

Note that MAPI stores all custom properties that represent date and time information using Greenwich Mean Time (GMT). The OLE Messaging Library converts these properties so that the values appear to the user in local time.

The OLE Messaging Library does not support MAPI properties of types PT_OBJECT and PT_CLSID. All others, however, are available through the Fields collection.

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
```

Count Property (Fields Collection)

The **Count** property returns the number of Field objects in the collection. Read-only.

Syntax

objFieldsColl.**Count**

Data Type

Long

Example

This code fragment 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
```

Delete Method (Fields Collection)

The **Delete** method deletes all user-defined fields in the Fields collection object.

Syntax

objFieldsColl.**Delete**

Parameters

objFieldsColl
Required. The Fields collection object.

Remarks

The **Delete** method deletes all user-defined fields and all fields considered optional by the underlying provider.

The **Delete** operation invalidates all the Field objects in the collection but does not remove them from memory. The programmer should set the invalidated objects to **Nothing** to remove them from memory, or reassign them to other fields. Attempted access to a deleted object results in a return of **mapiE_INVALID_OBJECT**.

Be cautious using the **Delete** method with a collection, because it deletes all the collection's member objects. To delete only one Field object, use the **Delete** method specific to that object.

The effect of the **Delete** method is not permanent until you use the **Update** method on the parent object containing the Fields collection, or the **Send** or **Delete** method if the parent is a Message object. Any member once permanently deleted cannot be recovered. However, the collection itself it still valid, and you can **Add** new members to it.

Item Property (Fields Collection)

The **Item** property returns a single Field object from the Fields collection. Read-only.

Syntax

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

objFieldsColl.**Item(***proptag***)**

objFieldsColl.**Item(***name* **[**, *propsetID***])**

objFieldsColl
Required. Specifies the Fields collection object.

index
Short integer (less than or equal to 65535 = 0xFFFF). Specifies the index within the collection.

proptag
Long integer (greater than or equal to 65536). Specifies the property tag value for the MAPI property to be retrieved.

name
String. Specifies the name of the user-defined property.

propsetID
Optional. String. Contains the unique identifier for the property set, represented as a string of hexadecimal characters. When *propsetID* is not supplied, the property set used for the access is the default property set value set by this collection's **SetNamespace** method, or the initial default property set value, PS_PUBLIC_STRINGS.

Data Type

Object

Remarks

The **Item** property works like an accessor property for small collections. In the Fields collection object it allows access to the predefined MAPI properties and to your own custom user-defined properties.

The *proptag* parameter in the second syntax contains the 32-bit MAPI property tag associated with the property and corresponds to the **ID** property of the Field object. The property tag contains the MAPI property identifier in its high-order 16 bits and the MAPI property type in its low-order 16 bits. All MAPI properties are accessible except those of MAPI type PT_OBJECT or PT_CLSID.

Several macros for C/C++ programmers are available in the MAPI SDK to help manipulate the MAPI 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 type and identifier components.

For example, you can use the following function to access a custom user-defined property using its property 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 the built-in MAPI properties are unnamed properties that can only be accessed using the numeric *proptag* value. They cannot be accessed using a string that represents the name. The following code fragment accesses 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 = mapiPR_MESSAGE_CLASS
    ' 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 " _
                         & "PR_MESSAGE_CLASS: type = " _
                         & objOneField.Type _
                         & "; value = " _
                         & objOneField.Value
        MsgBox strMsg
    End If
```

The OLE Messaging Library also supports multivalued MAPI properties.

You can also choose to access properties from other property sets, including your own, by either setting the *propsetID* parameter or by calling the **SetNamespace** method to set that property set's unique identifier.

The **Item** property is the default property of a Fields collection, meaning that *objFieldsColl*(*index*) is syntactically equivalent to *objFieldsColl*.**Item(***index***)** in Visual Basic code.

See Also

Customizing a Folder or Message, Viewing MAPI Properties

SetNamespace Method (Fields Collection)

The **SetNamespace** method selects the property set that is to be used for accessing MAPI named properties in the Fields collection.

Syntax

objFieldsColl.**SetNamespace** *PropsetID*

Parameters

objFieldsColl

Required. The Fields collection object.

PropsetID

Required. String. Contains a unique identifier that identifies the property set, represented as a string of hexadecimal characters. The *PropsetID* parameter identifies the property set to be used for subsequent named property accesses to a Field object in this Fields collection. An empty string resets the default to the property set PS_PUBLIC_STRINGS.

Remarks

Every MAPI named property belongs to a property set, each member of which uses the same **GUID** for the first part of its name. The set of all possible names within a property set is called its name space. The **SetNameSpace** method specifies which property set is to be in effect until changed by another call to this method. The MAPI named properties are accessed using the Fields collection's **Add** method and **Item** property.

The initial default value for the property set is PS_PUBLIC_STRINGS. To create your own property set for your named properties, supply a unique property set identifier to **SetNamespace**. This property set then replaces PS_PUBLIC_STRINGS as the default property set for all subsequent named property accesses using this object. The default property set is used unless explicitly overridden by the optional *PropsetID* parameter. The value is set only for the current object; to continue using the same property set for all objects, you must call **SetNamespace** for each object.

To define a new property set, obtain a string that contains hexadecimal characters representing a unique identifier. You can obtain this identifier using either the Win32 command-line utility **UUIDGEN** or by calling the Win32 function **CoCreateGuid**.

For more information on named properties and property sets, see "Named Properties" in the *MAPI Programmer's Reference*. For more information on **UUIDGEN** and **CoCreateGuid**, see the Win32 SDK documentation.

Folder Object

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

Quick Info

Specified in header file:	MDISP.ODL
First available in:	OLE Messaging Library version 1.0.a
Parent objects:	Folders collection InfoStore
Child objects:	Folder
Default property:	Messages

Properties

Name	Available in version	Type	Access
Application	1.0.a	String	Read-only
Class	1.0.a	Long	Read-only
Fields	1.0.a	Field object or Fields collection object	Read-only
FolderID	1.0.a	String	Read-only
Folders	1.0.a	Folders collection object	Read-only
ID	1.0.a	String	Read-only
MAPIOBJECT	1.0.a	Object	Read/write (Note: Not available to Visual Basic applications)
Messages	1.0.a	Messages collection object	Read-only
Name	1.0.a	String	Read/write
Parent	1.0.a	Folders collection object or InfoStore object	Read-only
Session	1.0.a	Session object	Read-only
StoreID	1.0.a	String	Read-only

Methods

Name	Available in version	Parameters
CopyTo	1.1	*folderID* as **String**, (optional) *storeID* as **String**, (optional) *name* as **String**, (optional) *copySubfolders* as

		Boolean
Delete	1.0.a	(none)
IsSameAs	1.1	*object* as **Object**
MoveTo	1.1	*folderID* as **String**, (optional) *storeID* as **String**
Update	1.0.a	(optional) *makePermanent* as **Boolean**, (optional) *refreshObject* as **Boolean**

Remarks

A Folder object is considered a top-level object, meaning it can be created directly from a Visual Basic program, through either late binding or early binding.

Changes to the folder are not saved by MAPI until you call its **Update** method.

CopyTo Method (Folder Object)

The **CopyTo** method makes a copy of the Folder object at another folder hierarchy location.

Syntax

Set *objCopiedFolder* = *objFolder*.**CopyTo(***folderID* **[***, storeID, name, copySubfolders* **])**

Parameters

objCopiedFolder
On successful return, contains the copied Folder object.

objFolder
Required. This Folder object.

folderID
Required. String. The unique identifier of the new parent Folder object, that is, the Folder object under which the copy of this folder is to appear as a subfolder.

storeID
Optional. String. The unique identifier of the InfoStore object in which the folder copy is to appear, if different from this folder's InfoStore.

name
Optional. String. The name to be assigned to the folder copy, if different from this folder's name.

copySubfolders
Optional. Boolean. If **True**, all subfolders contained within this folder are to be copied along with the folder.

Remarks

All Message objects contained within this folder are copied along with the folder itself. This also applies to messages contained in the subfolders if the *copySubfolders* parameter is **True**.

The copy operation takes effect immediately. This Folder object, together with all its contents, remains unchanged by the **CopyTo** method.

Delete Method (Folder Object)

The **Delete** method deletes the Folder object from its parent Folders collection or InfoStore object.

Syntax

objFolder.**Delete()**

Parameters

objFolder
Required. This Folder object.

Remarks

The action of the **Delete** method is permanent, and the Folder object cannot be recovered. Before calling the **Delete** method, the application can prompt the user to verify whether the folder should be permanently deleted.

The **Delete** operation invalidates the Folder object but does not remove it from memory. The programmer should set the invalidated object to **Nothing** to remove it from memory, or reassign it to another folder. Attempted access to a deleted object results in a return of **mapiE_INVALID_OBJECT**.

You can delete all the folders in the Folders collection by calling the collection's **Delete** method. The ability to delete any folder depends on the permissions granted to the user. The **Delete** method returns an error code if called with insufficient permissions.

Example

```
Function Folder_Delete()
    ' error handling here
    If objFolder Is Nothing Then
        MsgBox "must select a valid Folder object"
        Exit Function
    End If
    objFolder.Delete ()
    Set objFolder = Nothing
    Exit Function
End Function
```

Fields Property (Folder Object)

The **Fields** property returns a single Field object or a Fields collection object. Read-only.

Syntax

objFolder.**Fields**

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

objFolder.**Fields(***proptag***)**

objFolder.**Fields(***name***)**

index
Short integer (less than or equal to 65535). Specifies the index within the collection.

proptag
Long integer (greater than or equal to 65536). Specifies the property tag value for the MAPI property to be retrieved.

name
String. Specifies the name of the custom MAPI property.

Data Type

Object

Remarks

The **Fields** property returns one or more of the fields associated with a Folder object. Each field typically corresponds to a MAPI property.

The **Fields** property provides a generic access mechanism that allows Visual Basic and Visual C++ programmers to retrieve the value of a MAPI property using either its name or its MAPI property tag. For access with the property tag, use *objFolder*.**Fields(***proptag***)**, where *proptag* is the 32-bit MAPI property tag associated with the property, such as **mapiPR_MESSAGE_CLASS**. To access a named property, use *objFolder*.**Fields(***name***)**, where *name* is a string that represents the custom property name.

Example

This code fragment displays the field name or identifier value of all Field Object objects within the collection:

```
' many properties are MAPI properties and have no names
' for those properties, display the ID
' fragment from Field_Name
' assume objFieldColl, objOneField are valid objects
For i = 1 to objFieldColl.Count Step 1
    Set objOneField = objFieldColl.Index(i)
    If "" = objOneField.Name Then
        MsgBox "Field has no name; ID = " & objOneField.ID
    Else
        MsgBox "Field name = " & objOneField.Name
    End If
Next i
```

FolderID Property (Folder Object)

The **FolderID** property returns the unique identifier of the subfolder's parent folder as a string. Read-only.

Syntax

objFolder.**FolderID**

Data Type

String

Remarks

MAPI assigns a permanent, unique identifier when an object is created. This identifier does not change from one MAPI session to another, nor from one messaging domain to another.

Note that MAPI does not require identifier values to be binary comparable. Accordingly, two identifier values can be different, yet refer to the same object. You can compare identifiers using the MAPI **CompareEntryIDs** method. For more information, see the *MAPI Programmer's Reference*.

The **FolderID** property corresponds to the MAPI property PR_PARENT_ENTRYID, converted to a string of hexadecimal characters.

Example

```
'          fragment from Session_Inbox
    Set objFolder = objSession.Inbox
'          fragment from Folder_FolderID
    strFolderID = objFolder.FolderID
    MsgBox "Parent Folder ID = " & strFolderID
' later: obtain parent folder
'          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), **GetFolder** Method (Session Object), **StoreID** Property (Folder Object)

Folders Property (Folder Object)

The **Folders** property returns a Folders collection of subfolders within the folder. Read-only.

Syntax

objFolder.**Folders**

Data Type

Object

Example

This code fragment uses a recursive function to list the names of all subfolders of the specified folder:

```
' fragment from Session_Inbox
    Set objFolder = objSession.Inbox
' from TstDrv_Util_ListFolders
   If mapiFolder = objFolder.Class Then ' verify it's a 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
        Exit Function
    End If
    ' error handler here
End Function
```

ID Property (Folder Object)

The **ID** property returns the unique identifier of the Folder object as a string. Read-only.

Syntax

objFolder.**ID**

Data Type

String

Remarks

MAPI assigns a permanent, unique identifier when an object is created. This identifier does not change from one MAPI session to another, nor from one messaging domain to another.

The **ID** property corresponds to the MAPI property PR_ENTRYID, converted to a string of hexadecimal characters.

Example

```
' 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 "Current Folder ID = " & strFolderID
' later: restore folder 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

FolderID Property (Folder Object), **GetFolder** Method (Session Object), **StoreID** Property (Folder Object)

IsSameAs Method (Folder Object)

The **IsSameAs** method returns **True** if the Folder object is the same as the Folder object being compared against.

Syntax

objFolder.**IsSameAs(***objFolder2***)**

Parameters

objFolder
Required. This Folder object.

objFolder2
Required. The Folder object being compared against.

Remarks

Two Folder objects are considered to be the same if and only if they are instantiations of the same physical (persistent) object. Two objects with the same value are still considered different if they do not instantiate the same physical object, for example if one is a copy of the other. In such a case **IsSameAs** returns **False**.

The **IsSameAs** method ultimately calls one of the MAPI **CompareEntryIDs** methods to determine if two objects are the same. A generic comparison of any two object identifiers is also available with the Session object's **CompareIDs** method.

MAPIOBJECT Property (Folder Object)

The **MAPIOBJECT** property returns an **IUnknown** pointer to the Folder object. Not available to Visual Basic applications. Read/write.

Syntax

objFolder.**MAPIOBJECT**

Data Type

Variant (**vbDataObject** format)

Remarks

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 Introduction to Automation and How Programmable Objects Work. Also see the Microsoft *OLE Programmer's Reference*.

Messages Property (Folder Object)

The **Messages** property returns a Messages collection object within the folder. Read-only.

Syntax

objFolder.**Messages**

Data Type

Object

Remarks

The **Messages** property is the default property of a Folder object, meaning that *objFolder* is syntactically equivalent to *objFolder*.**Messages** in Visual Basic code.

Example

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

MoveTo Method (Folder Object)

The **MoveTo** method relocates the Folder object to another folder hierarchy location.

Syntax

Set *objMovedFolder* = *objFolder*.**MoveTo(***folderID* **[***, storeID* **])**

Parameters

objMovedFolder
On successful return, contains the moved Folder object.

objFolder
Required. This Folder object.

folderID
Required. String. The unique identifier of the new parent Folder object, that is, the Folder object under which this folder is to appear as a subfolder.

storeID
Optional. String. The unique identifier of the InfoStore object in which this folder is to appear, if different from its current InfoStore.

Remarks

All subfolders of this folder, together with all Message objects contained within this folder and its subfolders, are moved along with the folder itself.

The move operation takes effect immediately. This Folder object is no longer accessible at its former location after the **MoveTo** method returns.

Name Property (Folder Object)

The **Name** property returns or sets the name of the Folder object as a string. Read/write.

Syntax

objFolder.**Name**

Data Type

String

Remarks

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

Example

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

StoreID Property (Folder Object)

The **StoreID** property returns the unique identifier of the InfoStore object in which the Folder object resides. Read-only.

Syntax

objFolder.**StoreID**

Data Type

String

Remarks

MAPI assigns a permanent, unique identifier when an object is created. This identifier does not change from one MAPI session to another, nor from one messaging domain to another.

Note that MAPI does not require identifier values to be binary comparable. Accordingly, two identifier values can be different, yet refer to the same object. You can compare identifiers using the MAPI method **CompareEntryIDs**. For more information, see the *MAPI Programmer's Reference*.

The **StoreID** property corresponds to the MAPI property PR_STORE_ENTRYID, converted to a string of hexadecimal characters.

Example

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

See Also

FolderID Property (Folder Object), **GetFolder** Method (Session Object), **ID** Property (Folder Object)

Update Method (Folder Object)

The **Update** method saves changes to the Folder object in the MAPI system.

Syntax

objFolder.**Update([***makePermanent***, *refreshObject***])**

Parameters

objFolder
Required. The Folder object.

makePermanent
Optional. Boolean. A value of **True** indicates that the property cache is flushed and all changes are committed in the underlying message store. **False** indicates that the property cache is flushed but not committed to persistent storage. The default value is **True**.

refreshObject
Optional. Boolean. A value of **True** indicates that the property cache is reloaded from the values in the underlying message store. **False** indicates that the property cache is not reloaded. The default value is **False**.

Remarks

Changes to Folder objects are not permanently saved in the MAPI system until you call the **Update** method with the *makePermanent* parameter set to **True**.

For improved performance, the OLE Messaging Library caches property changes in private storage and updates either the object or the underlying persistent storage only when you explicitly request such an update. For efficiency, you should make only one call to **Update** with its *makePermanent* parameter set to **True**.

The *makePermanent* and *refreshObject* parameters combine to cause the following changes:

	refreshObject = True	**refreshObject = False**
makePermanent = True	Commit all changes, flush the cache, and reload the cache from the message store.	Commit all changes and flush the cache (default combination).
makePermanent = False	Flush the cache and reload the cache from the message store.	Flush the cache.

Call **Update(False, True)** to flush the cache and then reload the values from the message store.

Folders Collection Object

The Folders collection object contains one or more Folder objects.

Quick Info

Specified in header file:	MDISP.ODL
First available in:	OLE Messaging Library version 1.0.a
Parent objects:	Folder (Inbox or Outbox)
Child objects:	Folder
Default property:	Item

A Folders collection is considered a *large collection*, which means that the **Count** and **Item** properties have limited validity, and your best option is to use a Folder object identifier value or the **Get** methods to access an individual Folder object within the collection.

Properties

Name	Available in version	Type	Access
Application	1.0.a	String	Read-only
Class	1.0.a	Long	Read-only
Count	1.1	Long	Read-only
Item	1.1	Folder object	Read-only
Parent	1.0.a	Inbox or Outbox folder object	Read-only
Session	1.0.a	Session object	Read-only

Methods

Name	Available in version	Parameters
Add	1.1	*name* as String
Delete	1.1	(none)
GetFirst	1.0.a	(none)
GetLast	1.0.a	(none)
GetNext	1.0.a	(none)
GetPrevious	1.0.a	(none)
Sort	1.1	(optional) *SortOrder* as **Long**, (optional) *PropTag* as **Long**, (optional) *PropID* as **String**

Remarks

Large collections, such as the Folders collection, cannot always maintain an accurate count of the number of objects in the collection. It is strongly recommended that you use the **GetFirst**, **GetNext**, **GetLast**, and **GetPrevious** methods to access individual items in the collection. You can access one specific folder by using the Session object's **GetFolder** method, and you can access all the items in the collection with the Visual Basic **For Each** construction.

The order that items are returned by **GetFirst**, **GetNext**, **GetLast**, and **GetPrevious** depends on

whether the folders are sorted or not. The Folder objects within a collection can be sorted on a MAPI property of your choice, either ascending or descending, using the **Sort** method. When the items are not sorted, you should not rely on these methods to return the items in any specified order. The best programming approach to use with unsorted collections is to assume that the access functions are able to access all items within the collection, but that the order of the objects is not defined.

Example

To refer to a unique Folder object within the Folders collection, use the collection's **GetFirst** and **GetNext** methods or use the folder's **ID** value as an index.

The following code sample demonstrates the **Get** methods. The sample assumes that you have exactly three subfolders within your Inbox and exactly 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

Object Collections

Add Method (Folders Collection)

The **Add** method creates and returns a new Folder object in the Folders collection.

Syntax

Set *objFolder* = *objFoldersColl*.**Add(***name***)**

Parameters

objFolder
On successful return, contains the new Folder object.

objFoldersColl
Required. The Folders collection object.

name
Required. String. The display name of the folder.

Remarks

The *name* parameter corresponds to the **Name** property of the Folder object.

The user must have permission to **Add** or **Delete** a Folder object. Most users have this permission only for their personal folders.

The new Folder object is saved in the MAPI system when you call its **Update** method.

Example

This code fragment adds a new folder to a user's Inbox:

```
Dim myInbox, newFolder As Object
Set myInbox = MAPI.Session.Inbox
' add new folder to Inbox
Set newFolder = myInbox.Add "Personal Messages"
' commit new folder to collection
myInbox.Update
```

Count Property (Folders Collection)

The **Count** property returns the number of Folder objects in the collection, or a very large number if the exact count is not available. Read-only.

Syntax

objFoldersColl.**Count**

Data Type

Long

Remarks

The **Count** property is useful for determining whether a Folders collection is empty or not.

A large collection cannot always maintain an accurate count of its members, and the **Count** property cannot be used as the collection's size when it has a very large value such as **mapiMaxCount**. Programmers needing to access individual objects in a large collection are strongly advised to use the Visual Basic **For Each** statement or the **Get** methods.

The recommended procedures for traversing a large collection are, in decreasing order of preference:

1. Global selection, such as the Visual Basic **For Each** statement
2. The **Get** methods, particularly **GetFirst** and **GetNext**
1. An indexed loop, such as the Visual Basic **For ... Next** construction

If the message store provider cannot supply the precise number of Folder objects, the OLE Messaging Library returns a very large number for the **Count** property. On 32-bit platforms, this value is **mapiMaxCount**, which equals 2^31-1, or 2147483647. On other platforms, **mapiMaxCount** is not defined, and the OLE Messaging Library returns -1. A program on such a platform must ensure that -1 does not prematurely terminate any iteration based on the **Count** property.

Programmers using an indexed loop terminating on the **Count** property must also check each returned object for a value of **Nothing**. The loop must proceed forward from the beginning of the collection, and the index must have initial and increment values of 1. Results are undefined for any other procedure.

The use of the **Item** property in conjunction with the **Count** property in a large collection can be seen in the following example.

Example

This code fragment searches for a Folder object called "Resumes":

```
Dim i As Integer ' loop index / object counter
Dim collFolders as Object ' folders collection; assume already given
If collFolders Is Nothing Then
    ' MsgBox "Folders collection object is invalid"
    ' Exit
End If
' see if collection is empty
If 0 = collFolders.Count Then
    ' MsgBox "No folders in collection"
    ' Exit
End If
' look for folder called "Resumes" in collection
For i = 1 To collFolders.Count Step 1
    If collFolders.Item(i) Is Nothing Then
    ' MsgBox "No such folder found in collection"
```

```
    ' Exit ' no more folders in collection
    End If
    If collFolders.Item(i).Name = “Resumes” Then
    ' MsgBox "Desired folder is at index " & i
    ' Exit
    End If
Next i
```

Delete Method (Folders Collection)

The **Delete** method deletes all the folders in the Folders collection.

Syntax

objFoldersColl.**Delete()**

Parameters

objFoldersColl
Required. The Folders collection object.

Remarks

The **Delete** operation invalidates all the Folder objects in the collection but does not remove them from memory. The programmer should set the invalidated objects to **Nothing** to remove them from memory, or reassign them to other folders. Attempted access to a deleted object results in a return of **mapiE_INVALID_OBJECT**.

Be cautious using the **Delete** method with a collection, because it deletes all the collection's member objects. To delete only one Folder object, use the **Delete** method specific to that object.

The **Delete** method on a large collection takes effect immediately and is permanent. A deleted member cannot be recovered. However, the collection itself it still valid, and you can **Add** new members to it.

GetFirst Method (Folders Collection)

The **GetFirst** method returns the first Folder object in the Folders collection. It 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.

Remarks

The **Get** methods are similar to the **Find** and **Move** methods used with Microsoft Access, but they use a different syntax.

GetLast Method (Folders Collection)

The **GetLast** method returns the last Folder object in the Folders collection. It 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.

Remarks

The **Get** methods are similar to the **Find** and **Move** methods used with Microsoft Access, but they use a different syntax.

GetNext Method (Folders Collection)

The **GetNext** method returns the next Folder object in the Folders collection. It returns **Nothing** if no next object exists, for example if already positioned at the end of the 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.

Remarks

The **Get** methods are similar to the **Find** and **Move** methods used with Microsoft Access, but they use a different syntax.

GetPrevious Method (Folders Collection)

The **GetPrevious** method returns the previous Folder object in the Folders collection. It returns **Nothing** if no previous object exists, for example if already positioned at the beginning of 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.

Remarks

The **Get** methods are similar to the **Find** and **Move** methods used with Microsoft Access, but they use a different syntax.

Item Property (Folders Collection)

The **Item** property returns a single Folder object from the Folders collection. Read-only.

Syntax

objFoldersColl.**Item(***index***)**

objFoldersColl.**Item(***prefix***)**

index
A long integer ranging from 1 to the size of the Folders collection.

prefix
A string representing a prefix substring of a Folder object's **Name** property.

Data Type

Object

Remarks

The **Item** property is useful for satisfying syntax requirements when obtaining a member of a Folders collection.

A large collection cannot support true integer indexing, and the **Item(***index***)** syntax cannot be used for arbitrary selection of members of the collection. Programmers needing to access individual objects in a large collection are strongly advised to use the Visual Basic **For Each** statement or the **Get** methods, particularly **GetFirst** and **GetNext**.

The **Item(***index***)** syntax is provided solely as a placeholder in an indexed loop, such as the **For ... Next** construction in Visual Basic. Such a loop must proceed forward from the beginning of the collection, and the index must have initial and increment values of 1. Results are undefined for any other procedure.

For more information on using the **Count** and **Item** properties in a large collection, see the example in the **Count** property.

The **Item(***prefix***)** syntax returns the first Folder object whose **Name** property begins with the string specified by *prefix*.

The **Item** property is the default property of a Folders collection, meaning that *objFoldersColl***(***prefix***)** is syntactically equivalent to *objFoldersColl*.**Item(***prefix***)** in Visual Basic code.

Sort Method (Folders Collection)

The **Sort** method sorts the folders in the collection on the specified property according to the specified sort order.

Syntax

objFoldersColl.**Sort([***SortOrder, PropTag***])**

objFoldersColl.**Sort([***SortOrder, PropID***])**

Parameters

objFoldersColl

Required. The Folders collection object.

SortOrder

Optional. Long. The specified sort order, one of the following values:

Value	Numeric value	Description
mapiNone	0	No sort
mapiAscending	1	Ascending sort (default)
mapiDescending	2	Descending sort

PropTag

Optional. Long. The property tag value for the MAPI property to be used for the sort. *PropTag* is the 32-bit MAPI property tag associated with the property, such as **mapiPR_STORE_ENTRYID**.

PropID

Optional. String. The custom property name of a MAPI named property.

Remarks

Both parameters are optional. If *SortOrder* is not specified, ascending order is used. If neither *PropTag* nor *PropID* is specified, the property used in the previous call to **Sort** is used again. If **Sort** has never been called on this collection during this session, the MAPI property **mapiPR_DISPLAY_NAME** is used for the sort.

InfoStore Object

The InfoStore object provides access to the folder hierarchy of a message store.

Quick Info

Specified in header file:	MDISP.ODL
First available in:	OLE Messaging Library version 1.0.a
Parent objects:	InfoStores collection
Child objects:	Folder
Default property:	Name

Properties

Name	Available in version	Type	Access
Application	1.0.a	String	Read-only
Class	1.0.a	Long	Read-only
ID	1.0.a	String	Read-only
Index	1.0.a	Long	Read-only
Name	1.0.a	String	Read-only
Parent	1.0.a	InfoStores collection object	Read-only
ProviderName	1.0.a	String	Read-only
RootFolder	1.0.a	Folder object	Read-only
Session	1.0.a	Session object	Read-only

Methods

Name	Available in version	Parameters
IsSameAs	1.1	*object* as **Object**

Remarks

The InfoStore object provides access to its interpersonal message folder hierarchy through the **RootFolder** property, which returns the Folder object that represents the root of the IPM subtree. To access the root folder of the entire message store, first obtain its identifier with the **FolderID** property of the IPM root folder, and then call the Session object's **GetFolder** method.

You can obtain any InfoStore object available to this session with the **Item** property of the InfoStores collection. You can also retrieve an InfoStore object with a known identifier by calling the session's **GetInfoStore** method.

Example

```
Dim objInfoStore, objIPMRoot, objStoreRoot as Object
Dim rootID as String
Set objInfoStore = objSession.InfoStores.Item(1)
Set objIPMRoot = objInfoStore.RootFolder
rootID = objIPMRoot.FolderID
Set objStoreRoot = objSession.GetInfoStore (rootID)
```

ID Property (InfoStore Object)

The **ID** property returns the unique identifier of the InfoStore object as a string. Read-only.

Syntax

objInfoStore.**ID**

Data Type

String

Remarks

MAPI systems assign a permanent, unique identifier when an object is created. This identifier does not change from one MAPI session to another, nor from one messaging domain to another. The InfoStore identifier can be used in subsequent calls to the Session object's **GetInfoStore** method.

The **ID** property corresponds to the MAPI property PR_ENTRYID, converted to a string of hexadecimal characters.

Example

```
Dim strInfoStoreID as String  ' hex string version of ID
Dim objInfoStore as Object    ' assume valid
    strInfoStoreID = objInfoStore.ID ' global variable
    MsgBox "InfoStore ID = " & strInfoStoreID
' ... this ID can be used as the parameter to the Session method
    Set objInfoStore = objSession.GetInfoStore(strInfoStoreID)
```

Index Property (InfoStore Object)

The **Index** property returns the index number for the InfoStore object within the parent InfoStores collection. Read-only.

Syntax

objInfoStore.**Index**

Data Type

Long

Remarks

The **Index** property indicates this object's position within the parent InfoStores collection. It can be used later as the *index* parameter to the collection's **Item** property to access this message store again.

Example

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

IsSameAs Method (InfoStore Object)

The **IsSameAs** method returns **True** if the InfoStore object is the same as the InfoStore object being compared against.

Syntax

objInfoStore.**IsSameAs**(*objInfoStore2*)

Parameters

objInfoStore
Required. This InfoStore object.

objInfoStore2
Required. The InfoStore object being compared against.

Remarks

Two InfoStore objects are considered to be the same if and only if they are instantiations of the same physical (persistent) object. Two objects with the same value are still considered different if they do not instantiate the same physical object, for example if one is a copy of the other. In such a case **IsSameAs** returns **False**.

The **IsSameAs** method ultimately calls one of the MAPI **CompareEntryIDs** methods to determine if two objects are the same. A generic comparison of any two object identifiers is also available with the Session object's **CompareIDs** method.

Name Property (InfoStore Object)

The **Name** property returns the name of the InfoStore object as a string. Read-only.

Syntax

objInfoStore.**Name**

Data Type

String

Remarks

The string “Public Folders” is the name of the InfoStore object that contains the public folders.

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

The **Name** property is the default property of an InfoStore object, meaning that *objInfoStore* is syntactically equivalent to *objInfoStore*.**Name** in Visual Basic code.

Example

```
Dim objInfoStore As Object  ' assume valid InfoStore object
MsgBox "InfoStore name = " & objInfoStore.Name
```

ProviderName Property (InfoStore Object)

The **ProviderName** property returns the name of the InfoStore's message store provider as a string. Read-only.

Syntax

objInfoStore.**ProviderName**

Data Type

String

Remarks

A message store provider is a MAPI object that manages one or more MAPI message stores. Each message store is accessibe as an OLE Messaging Library InfoStore object.

The **ProviderName** property corresponds to the MAPI property PR_PROVIDER_DISPLAY.

Example

```
Dim objInfoStore As Object ' assume valid InfoStore object
MsgBox "Message store provider name = " & objInfoStore.ProviderName
```

RootFolder Property (InfoStore Object)

The **RootFolder** property returns a Folder object representing the root of the IPM subtree for the InfoStore object. Read-only.

Syntax

Set *objFolder* **=** *objInfoStore*.**RootFolder**

Data Type

Object (Folder object)

Remarks

The **RootFolder** property provides a convenient way to get to this commonly used Folder object.

In addition to the general ability to navigate through the formal collection and object hierarchy, the OLE Messaging Library supports properties that allow your application to directly access the most common Folder objects:

- The InfoStore object's **RootFolder** property for the IPM subtree root folder
- The Session object's **Inbox** property for the Inbox folder
- The Session object's **Outbox** property for the Outbox folder

Some message stores also support a direct way to obtain the root folder of the message store. For more information, see the Session object's **GetFolder** method.

Example

```
' from InfoStores_RootFolder
    If objInfoStore Is Nothing Then
        MsgBox "must first select an InfoStore object"
        Exit Function
    End If
    Set objFolder = objInfoStore.RootFolder
    If objFolder Is Nothing Then
        MsgBox "Unable to retrieve IPM root folder"
        Set objMessages = Nothing
        Exit Function
    End If
    If objFolder.Name = "" Then
        MsgBox "Folder set to folder with no name, ID = " _
                                                & objFolder.ID
    Else
        MsgBox "Folder set to: " & objFolder.Name
    End If
    Set objMessages = objFolder.Messages
    Exit Function
```

InfoStores Collection Object

The InfoStores collection object contains one or more InfoStore objects.

Quick Info

Specified in header file:	MDISP.ODL
First available in:	OLE Messaging Library version 1.0.a
Parent objects:	Session
Child objects:	InfoStore
Default property:	Item

An InfoStores collection is considered a *small collection*, which means that it supports count and index values that let you access an individual InfoStore object through the **Item** property. The InfoStores collection supports the Visual Basic **For Each** statement.

Properties

Name	Available in version	Type	Access
Application	1.0.a	String	Read-only
Class	1.0.a	Long	Read-only
Count	1.0.a	Long	Read-only
Item	1.0.a	InfoStore object	Read-only
Parent	1.0.a	Session object	Read-only
Session	1.0.a	Session object	Read-only

Methods

(None.)

Remarks

An InfoStores collection provides access to all InfoStore objects available to this session. Each InfoStore object in turn offers access to the folder hierarchy of that message store. This is used primarily to obtain access to the public folders.

The OLE Messaging Library does not support methods to add or remove InfoStore objects from the collection.

In general, you cannot assume that the InfoStore object's **Name** property is unique. This means that you cannot rely on the name to retrieve the InfoStore from the collection. However, you can iterate through all objects in the collection using the InfoStores collection object's **Item** property, and then examine properties of the individual InfoStore objects. You can also rely on the InfoStore object's **ID** property, which is guaranteed to be unique.

Count Property (InfoStores Collection)

The **Count** property returns the number of InfoStore objects in the collection. Read-only.

Syntax

objInfoStoresColl.**Count**

Data Type

Long

Example

This code fragment maintains a global variable to loop through the small collection, and uses the **Count** property to keep it from getting too large:

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

Item Property (InfoStores Collection)

The **Item** property returns a single InfoStore object from the InfoStores collection. Read-only.

Syntax

objInfoStoresColl.**Item(***index***)**

index
A long integer that ranges from 1 to *objInfoStoresColl*.**Count**.

Data Type

Object

Remarks

The **Item** property works like an accessor property for small collections.

The **Item** property is the default property of an InfoStores collection, meaning that *objInfoStoresColl***(***index***)** is syntactically equivalent to *objInfoStoresColl*.**Item(***index***)** in Visual Basic code.

Example

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

Message Object

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

Quick Info

Specified in header file:	MDISP.ODL
First available in:	OLE Messaging Library version 1.0.a
Parent objects:	Messages collection
Child objects:	Attachments collection Fields collection Recipients collection
Default property:	Subject

Properties

Name	Available in version	Type	Access
Application	1.0.a	String	Read-only
Attachments	1.0.a	Attachments collection object	Read-only
Class	1.0.a	Long	Read-only
Conversation	1.0.a	(Obsolete. Do not use.)	Read/write
ConversationIndex	1.0.a	String	Read/write
ConversationTopic	1.0.a	String	Read/write
DeliveryReceipt	1.0.a	Boolean	Read/write
Encrypted	1.0.a	Boolean	Read/write
Fields	1.0.a	Field object or Fields collection object	Read-only
FolderID	1.0.a	String	Read-only
ID	1.0.a	String	Read-only
Importance	1.0.a	Long	Read/write
MAPIOBJECT	1.0.a	(Not for use with Visual Basic.)	Read/write (Note: Not available to Visual Basic applications)
Parent	1.0.a	Attachments collection object, Fields collection object, or Recipients collection object	Read-only
ReadReceipt	1.0.a	Boolean	Read/write
Recipients	1.0.a	Recipients object or Recipients collection	Read/write

		object	
Sender	1.0.a	AddressEntry object	Read-only
Sent	1.0.a	Boolean	Read/write
Session	1.0.a	Session object	Read-only
Signed	1.0.a	Boolean	Read/write
Size	1.0.a	Long	Read-only
StoreID	1.0.a	String	Read-only
Subject	1.0.a	String	Read/write
Submitted	1.0.a	Boolean	Read/write
Text	1.0.a	String	Read/write
TimeReceived	1.0.a	Variant (**vbDate** format)	Read/write
TimeSent	1.0.a	Variant (**vbDate** format)	Read/write
Type	1.0.a	String	Read/write
Unread	1.0.a	Boolean	Read/write

Methods

Name	Available in version	Parameters
CopyTo	1.1	*folderID* as **String**, (optional) *storeID* as **String**
Delete	1.0.a	(none)
IsSameAs	1.1	*object* as **Object**
MoveTo	1.1	*folderID* as **String**, (optional) *storeID* as **String**
Options	1.0.a	(optional) *parentWindow* as **Long**
Send	1.0.a	(optional) *saveCopy* as **Boolean**, (optional) *showDialog* as **Boolean**, (optional) *parentWindow* as **Long**
Update	1.0.a	(optional) *makePermanent* as **Boolean**, (optional) *refreshObject* as **Boolean**

Remarks

A Message object is considered a top-level object, meaning it can be created directly from a Visual Basic program, through either late binding or early binding.

Visual Basic programmers can create new message objects using the Messages collection's **Add** method.

C/C++ programmers can create new message objects using the OLE function **CoCreateInstance**.

See Also

GetMessage Method (Session Object), **Messages** Property (Folder Object)

Attachments Property (Message Object)

The **Attachments** property returns a single Attachment object or an Attachments collection collection object. Read-only.

Syntax

Set *objAttachColl* **=** *objMessage.***Attachments**

Set *objOneAttach* = *objMessage.***Attachments(***index***)**

objAttachColl
Object. An Attachments collection object.

objMessage
Object. The Message object.

objOneAttach
Object. A single Attachment object.

index
Long. Specifies the number of the attachment within the Attachments collection. Ranges from 1 to the value specified by the Attachments collection's **Count** property.

Remarks

You can change individual Attachment objects within the Attachments collection, **Add** them to the collection, and **Delete** them from the collection.

Example

This code fragment uses the **Attachments** property to retrieve an attachment of the 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)
```

Conversation Property (Message Object)

The **Conversation** property is obsolete. It has been replaced by the **ConversationIndex** and **ConversationTopic** properties.

For more information on conversations, see Working With Conversations.

ConversationIndex Property (Message Object)

The **ConversationIndex** property specifies the index to the conversation thread of the message. Read/write.

Syntax

objMessage.**ConversationIndex**

Data Type

String

Remarks

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 order the messages within the 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 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 Server 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.

For more information on conversations, see Working With Conversations.

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

Example

This code fragment takes advantage of an OLE 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" ' zero time stamp
    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."
        Set objOneRecip = .Recipients.Add(Name:="Car Ads", Type:=mapiTo)
        ' or you could pick the public folder from the address book
        If objOneRecip Is Nothing Then
            MsgBox "Unable to create the public folder recipient"
            Exit Function
        End If
        .Recipients.Resolve
        .Update ' save everything in the MAPI system
        .Send showDialog:=False
    End With
End Function
```

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

```
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() ' new stamp
        ' 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 ' save everything in the MAPI system
        .Send showDialog:=False
    End With
' ... error handling
End Function
```

ConversationTopic Property (Message Object)

The **ConversationTopic** property specifies the subject of the conversation thread of the message. Read/write.

Syntax

objMessage.**ConversationTopic**

Data Type

String

Remarks

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's **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 that of 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.

For more information on conversations, see Working With Conversations.

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

Example

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

CopyTo Method (Message Object)

The **CopyTo** method makes a copy of the Message object in another folder.

Syntax

Set *objCopiedMessage* = *objMessage*.**CopyTo(***folderID* **[***, storeID***])**

Parameters

objCopiedMessage
On successful return, contains the copied Message object.

objMessage
Required. This Message object.

folderID
Required. String. The unique identifier of the destination Folder object in which the copy of this message is to appear.

storeID
Optional. String. The unique identifier of the InfoStore object in which the message copy is to appear, if different from this message's InfoStore.

Remarks

All properties that have been set on this message are copied, whether they have read-only or read/write access. Each property is copied with its value and access unchanged.

The copy operation takes effect when you call the **Update** method on the copied Message object. This allows you to change, for example, the **Sent** property on the message copy before committing the transaction.

This Message object remains unchanged by the **CopyTo** method.

Delete Method (Message Object)

The **Delete** method deletes the Message object.

Syntax

objMessage.**Delete**

Parameters

objMessage
Required. The Message object.

Remarks

The action of the **Delete** method is permanent, and the Message object cannot be recovered. Before calling the **Delete** method, the application can prompt the user to verify whether the message should be permanently deleted.

The **Delete** operation invalidates the Message object but does not remove it from memory. The programmer should set the invalidated object to **Nothing** to remove it from memory, or reassign it to another message. Attempted access to a deleted object results in a return of **mapiE_INVALID_OBJECT**.

You can delete all the messages in the Messages collection by calling the collection's **Delete** method. The ability to delete any message depends on the permissions granted to the user. The **Delete** method returns an error code if called with insufficient permissions.

DeliveryReceipt Property (Message Object)

The **DeliveryReceipt** property is **True** if a delivery-receipt notification message is requested. Read/write.

Syntax

objMessage.**DeliveryReceipt**

Data Type

Boolean

Remarks

Set the **DeliveryReceipt** property to **True** to obtain a notification message when the recipients receive your message. The default setting for the OLE Messaging Library is **False**.

Each transport provider that handles your message sends you a single delivery notification containing the names and addresses of all recipients it was delivered to. Note that delivery does not imply that the message has been read.

Notification requests include the **DeliveryReceipt** and **ReadReceipt** properties. For more information, see Making Sure The Message Gets There.

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

Encrypted Property (Message Object)

The **Encrypted** property is **True** if the message has been encrypted. Read/write.

Syntax

objMessage.**Encrypted**

Data Type

Boolean

Remarks

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

Security features include the **Encrypted** and **Signed** properties. For more information, see Securing Messages.

The **Encrypted** property corresponds to the SECURITY_ENCRYPTED flag of the MAPI property PR_SECURITY.

Fields Property (Message Object)

The **Fields** property returns a single Field object or a Fields collection object. Read-only.

Syntax

objMessage.**Fields**

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

objMessage.**Fields(***proptag***)**

objMessage.**Fields(***name***)**

index
Short integer (less than or equal to 65535). Specifies the index within the Fields collection.

proptag
Long integer (greater than or equal to 65536). Specifies the property tag value for the MAPI property to be retrieved.

name
String. Specifies the name of the custom MAPI property.

Data Type

Object

Remarks

The **Fields** property returns one or more of the fields associated with a Message object. Each field typically corresponds to a MAPI property.

The **Fields** property provides a generic access mechanism that allows Visual Basic and Visual C++ programmers to retrieve the value of a MAPI property using either its name or its MAPI property tag. For access with the property tag, use *objMessage*.**Fields**(*proptag*), where *proptag* is the 32-bit MAPI property tag associated with the property, such as **mapiPR_MESSAGE_CLASS**. To access a named property, use *objMessage*.**Fields**(*name*), where *name* is a string that represents the custom property name.

Example

```
' get the message's Fields collection
    Set objFieldsColl = objOneMsg.Fields
' get the first field of the Fields collection of the message
    i = 1
    Set objOneField = objFieldsColl.Item(i)
    If objOneField Is Nothing Then
        MsgBox "error; cannot get this Field object"
    Else
        MsgBox "Selected Field " & i
    End If
```

FolderID Property (Message Object)

The **FolderID** property returns the unique identifier of the folder in which the message resides. Read-only.

Syntax

objMessage.**FolderID**

Data Type

String

Remarks

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

MAPI assigns a permanent, unique identifier when an object is created. This identifier does not change from one MAPI session to another, nor from one messaging domain to another.

The **FolderID** property corresponds to the MAPI property PR_PARENT_ENTRYID, converted to a string of hexadecimal characters.

ID Property (Message Object)

The **ID** property returns the unique identifier of the Message object as a string. Read-only.

Syntax

objMessage.**ID**

Data Type

String

Remarks

The **ID** property can be used to retrieve this message at a later time, using the Session object's **GetMessage** method.

MAPI assigns a permanent, unique identifier when an object is created. This identifier does not change from one MAPI session to another, nor from one messaging domain to another.

The **ID** property corresponds to the MAPI property PR_ENTRYID, converted to a string of hexadecimal characters.

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)
```

Importance Property (Message Object)

The **Importance** property returns or sets the importance of the message as **mapiNormal** (the default), **mapiLow**, or **mapiHigh**. Read/write.

Syntax

objMessage.**Importance**

Data Type

Long

Remarks

The following values are defined:

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

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

Example

This code fragment sets the importance of a message as high:

```
' from the sample function QuickStart:
    Set objMessage = objSession.Outbox.Messages.Add
    ' ... check 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)

IsSameAs Method (Message Object)

The **IsSameAs** method returns **True** if the Message object is the same as the Message object being compared against.

Syntax

objMessage.**IsSameAs(***objMessage2***)**

Parameters

objMessage
Required. This Message object.

objMessage2
Required. The Message object being compared against.

Remarks

Two Message objects are considered to be the same if and only if they are instantiations of the same physical (persistent) object. Two objects with the same value are still considered different if they do not instantiate the same physical object, for example if one is a copy of the other. In such a case **IsSameAs** returns **False**.

The **IsSameAs** method ultimately calls one of the MAPI **CompareEntryIDs** methods to determine if two objects are the same. A generic comparison of any two object identifiers is also available with the Session object's **CompareIDs** method.

MAPIOBJECT Property (Message Object)

The **MAPIOBJECT** property returns an **IUnknown** pointer to the Message object. Not available to Visual Basic applications. Read/write.

Syntax

objMessage.**MAPIOBJECT**

Data Type

Variant (**vbDataObject** format)

Remarks

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 Introduction to Automation and How Programmable Objects Work. Also see the Microsoft *OLE Programmer's Reference*.

MoveTo Method (Message Object)

The **MoveTo** method relocates the Message object to another folder.

Syntax

Set *objMovedMessage* = *objMessage*.**MoveTo(***folderID* **[**, *storeID* **])**

Parameters

objMovedMessage
On successful return, contains the moved Message object.

objMessage
Required. This Message object.

folderID
Required. String. The unique identifier of the destination Folder object in which this message is to appear.

storeID
Optional. String. The unique identifier of the InfoStore object in which the message is to appear, if different from this current InfoStore.

Remarks

All properties that have been set on this message are moved, whether they have read-only or read/write access. Each property is moved with its value and access unchanged.

The move operation takes effect immediately. This Message object is no longer accessible at its former location after the **MoveTo** method returns.

Options Method (Message Object)

The **Options** method displays a modal dialog box where the user can change the submission options for a message.

Syntax

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

Parameters

objMessage
Required. The Message object.

parentWindow
Optional. Long. The parent window handle for the options dialog box. A value of zero (the default) specifies that the dialog should be application-modal.

Remarks

The **Options** dialog is always modal, meaning the parent window is disabled while the dialog is active. If the *parentWindow* parameter is set to zero or is not set, all windows belonging to the application are disabled while the dialog is active. If the *parentWindow* parameter is supplied but is not valid, the call returns **mapiE_INVALID_PARAMETER**.

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 **Options** method returns the error code **mapiE_NOT_FOUND**.

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

ReadReceipt Property (Message Object)

The **ReadReceipt** property is **True** if a read-receipt notification message is requested. Read/write.

Syntax

objMessage.**ReadReceipt**

Data Type

Boolean

Remarks

Set the **ReadReceipt** property to **True** to obtain a notification message when each recipient reads your message. The default setting for the OLE Messaging Library is **False**.

Each message store that receives your message sends you an individual read notification each time one of the recipients sets the read flag on the message. Note that the read flag being set does not imply that the recipient has physically read the message. Move and copy operations, for example, typically set the read flag.

Notification requests include the **DeliveryReceipt** and **ReadReceipt** properties. For more information, see Making Sure The Message Gets There.

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

Recipients Property (Message Object)

The **Recipients** property returns a single Recipient object or a Recipients collection object. Read/write.

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's **Count** property.

Data Type

Object

Remarks

You can change individual Recipient objects within the Recipients collection, **Add** them to the collection, and **Delete** them from the collection. You can also manipulate the Recipients collection as a whole with a single Visual Basic instruction. For example, you can copy the complete recipient list of a received message, with all their properties, to a reply message:

```
Set objReplyMessage.Recipients = objReceivedMessage.Recipients
```

Note that the **Attachments** property cannot be copied as a whole; attachments must be dealt with in the manner of the following example.

Example

This code fragment copies each of the recipients from the original message *objOneMsg* to the copy *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 ' now copy the name
        End If
    Next i
```

Send Method (Message Object)

The **Send** method sends the message to the recipients through the MAPI system.

Syntax

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

Parameters

objMessage
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 Items folder.

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

parentWindow
Optional. Long. The parent window handle for the **Send Message** dialog box. A value of zero (the default) specifies that the dialog should be application-modal. The *parentWindow* parameter is ignored unless *showDialog* is **True**.

Remarks

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 composed 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 Items folder.

The **Send** dialog is always modal, meaning the parent window is disabled while the dialog is active. If the *parentWindow* parameter is set to zero or is not set, all windows belonging to the application are disabled while the dialog is active. If the *parentWindow* parameter is supplied but is not valid, the call returns **mapiE_INVALID_PARAMETER**.

Note that even if *showDialog* is set to **True**, the OLE Messaging Library does not display the dialog box if the recipient has a null display name. The dialog box is, however, displayed for a null recipient (when the Recipient object is set to **Nothing**).

For more information on sending and posting messages, see Creating and Sending a Message and Posting Messages to a Public Folder.

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

See Also

Sent Property (Message Object), **Submitted** Property (Message Object)

Sender Property (Message Object)

The **Sender** property returns the sender of a message as an AddressEntry object. Read-only.

Syntax

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

objAddrEntry
Object. The returned AddressEntry object that represents the messaging user that sent the message.

objMessage
Object. The Message object.

Data Type

Object

Remarks

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

Example

This code fragment 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)

The **Sent** property is **True** if the message has been sent through the MAPI system. Read/write.

Syntax

objMessage.**Sent**

Data Type

Boolean

Remarks

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

For all three kinds of message, 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
Sent	**Send**	**Send** method sets **True**	Spooler sets **True**	Spooler sets **True**
Posted	**Update**	Application sets **False**	Application sets **True**	Application sets **True**
Saved	**Update**	Application sets **False**	Application sets **False**	Application sets **True**

For sent messages, the **Sent** property can be written 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.

For posted messages, 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**.

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.

For more information on sending and posting messages, see Creating and Sending a Message and Posting Messages to a Public Folder.

The **Sent** property corresponds to the MSGFLAG_UNSENT flag not being set in the MAPI property

PR_MESSAGE_FLAGS.

Signed Property (Message Object)

The **Signed** property is **True** if the message has been tagged with a digital signature. Read/write.

Syntax

objMessage.**Signed**

Data Type

Boolean

Remarks

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

Security features include the **Encrypted** and **Signed** properties. For more information, see Securing Messages.

The **Signed** property corresponds to the SECURITY_SIGNED flag of the MAPI property PR_SECURITY.

Size Property (Message Object)

The **Size** property returns the approximate size in bytes of the message. Read-only.

Syntax

objMessage.**Size**

Data Type

Long

Remarks

The **Size** property contains the sum, in bytes, of the sizes of all properties on this Message object, including in particular the **Attachments** property. It can be considerably greater than the size of the **Text** property alone.

The **Size** property is computed by the message store and is not valid until after the first **Update** or **Send** operation. Note that not all message stores support this property.

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

StoreID Property (Message Object)

The **StoreID** property represents the unique identifier for the message store that contains the message. Read-only.

Syntax

objMessage.**StoreID**

Data Type

String

Remarks

You can save the **ID** and **StoreID** properties of this message in order to recall it later with the Session object's **GetMessage** method.

The **StoreID** property corresponds to the MAPI property PR_STORE_ENTRYID, converted to a string of hexadecimal characters.

Subject Property (Message Object)

The **Subject** property returns or sets the subject of the message as a string. Read/write.

Syntax

objMessage.**Subject**

Data Type

String

Remarks

In a conversation thread, the **Subject** property is often used to set the **ConversationTopic** property.

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

The **Subject** property is the default property of a Message object, meaning that *objMessage* is syntactically equivalent to *objMessage*.**Subject** in Visual Basic code.

Example

This code fragment 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)

The **Submitted** property is **True** when the message has been submitted. Read/write.

Syntax

objMessage.**Submitted**

Data Type

Boolean

Remarks

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

For all three kinds of message, 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
Sent	**Send**	**Send** method sets **True**	Spooler sets **True**	Spooler sets **True**
Posted	**Update**	Application sets **False**	Application sets **True**	Application sets **True**
Saved	**Update**	Application sets **False**	Application sets **False**	Application sets **True**

For sent messages, the **Sent** property can be written 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.

For posted messages, 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**.

For more information on sending and posting messages, see Creating and Sending a Message and Posting Messages to a Public Folder.

The **Submitted** property corresponds to the MSGFLAG_SUBMIT flag not being set in the MAPI property PR_MESSAGE_FLAGS.

Text Property (Message Object)

The **Text** property returns or sets the text of the message as a string. Read/write.

Syntax

objMessage.**Text**

Data Type

String

Remarks

The message text is the principal content of an interpersonal message, typically displayed to each recipient as an immediate result of opening the message. The **Text** property specifically excludes various other message properties such as **Subject**, **Attachments**, and **Recipients**.

Note that the **Text** property is a plain text representation of the message text and does not support formatted text.

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

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

Example

This code fragment sets the text of a message:

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

TimeReceived Property (Message Object)

The **TimeReceived** property sets or returns the date and time the message was received as a **vbDate** variant data type. Read/write.

Syntax

objMessage.**TimeReceived**

Data Type

Variant (**vbDate** format)

Remarks

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's **Send** method, MAPI 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's **Item** property, the time values represent Greenwich Mean Time.

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

Example

This code fragment 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
```

TimeSent Property (Message Object)

The **TimeSent** property sets or returns the date and time the message was sent as a **vbDate** variant data type. Read/write.

Syntax

objMessage.**TimeSent**

Data Type

Variant (**vbDate** format)

Remarks

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's **Send** method, MAPI 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's **Item** property, the time values represent Greenwich Mean Time.

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

Example

This code fragment 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
```

Type Property (Message Object)

The **Type** property returns or sets the MAPI message class for the message. Read/write.

Syntax

objMessage.**Type**

Data Type

String

Remarks

The **Type** property contains the MAPI message class, which determines the set of properties defined for the message, the kind of information it conveys, and how it is to be handled. The message class consists of ASCII strings concatenated with periods, each string representing a level of subclassing. A standard interpersonal message has message class IPM.Note, which is a subclass of IPM and a superclass of IPM.Note.Private.

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

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. By default, the OLE Messaging Library sets the **Type** value of new messages to the MAPI message class IPM.Note.

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

Unread Property (Message Object)

The **Unread** property is **True** if the message has not been read by the current user. Read/write.

Syntax

objMessage.**Unread**

Data Type

Boolean

Remarks

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

For all three kinds of message, 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
Sent	**Send**	**Send** method sets **True**	Spooler sets **True**	Spooler sets **True**
Posted	**Update**	Application sets **False**	Application sets **True**	Application sets **True**
Saved	**Update**	Application sets **False**	Application sets **False**	Application sets **True**

For sent messages, the **Sent** property can be written 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.

For posted messages, 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**.

For more information on sending and posting messages, see Creating and Sending a Message and Posting Messages to a Public Folder.

The **Unread** property corresponds to the MSGFLAG_READ flag not being set in the MAPI property PR_MESSAGE_FLAGS.

Update Method (Message Object)

The **Update** method saves the message in the MAPI system.

Syntax

objMessage.**Update([***makePermanent***, *refreshObject*])**

Parameters

objMessage
Required. The Message object.

makePermanent
Optional. Boolean. A value of **True** indicates that the property cache is flushed and all changes are committed in the underlying message store. **False** indicates that the property cache is flushed but not committed to persistent storage. The default value is **True**.

refreshObject
Optional. Boolean. A value of **True** indicates that the property cache is reloaded from the values in the underlying message store. **False** indicates that the property cache is not reloaded. The default value is **False**.

Remarks

Changes to Message objects are not permanently saved in the MAPI system until you call the **Update** method with the *makePermanent* parameter set to **True**.

For improved performance, the OLE Messaging Library caches property changes in private storage and updates either the object or the underlying persistent storage only when you explicitly request such an update. For efficiency, you should make only one call to **Update** with its *makePermanent* parameter set to **True**.

The *makePermanent* and *refreshObject* parameters combine to cause the following changes:

	refreshObject = True	**refreshObject = False**
makePermanent = True	Commit all changes, flush the cache, and reload the cache from the message store.	Commit all changes and flush the cache (default combination).
makePermanent = False	Flush the cache and reload the cache from the message store.	Flush the cache.

Call **Update(False, True)** to flush the cache and then reload the values from the message store.

Example

This code fragment changes the subject of the first message in the Inbox:

```
Set objMessage = objSession.Inbox.GetFirst
' ... verify message
objMessage.Subject = "This is the new subject"
objMessage.Update ' commit changes to MAPI system
```

To add a new Message object, use the Messages collection's **Add** method followed by the message's **Update** method. This code fragment saves a new message in the Outbox:

```
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 makePermanent:=True ' redundant parameter (default)
```

See Also

Send Method (Message Object)

MessageFilter Object

The MessageFilter object specifies criteria for restricting a search on a Messages collection.

Quick Info

Specified in header file:	MDISP.ODL
First available in:	OLE Messaging Library version 1.1
Parent objects:	Messages collection
Child objects:	Fields collection
Default property:	Subject

Properties

Name	Available in version	Type	Access
Application	1.1	String	Read-only
Class	1.1	Long	Read-only
Conversation	1.1	String	Read/write
Fields	1.1	Field object or Fields collection object	Read-only
Importance	1.1	Long	Read/write
Not	1.1	Boolean	Read/write
Or	1.1	Boolean	Read/write
Parent	1.1	Messages collection object	Read-only
Recipients	1.1	Recipient object or Recipients collection object	Read/write
Sender	1.1	AddressEntry object	Read/write
Sent	1.1	Boolean	Read/write
Session	1.1	Session object	Read-only
Size	1.1	Long	Read/write
Subject	1.1	String	Read/write
Text	1.1	String	Read/write
TimeFirst	1.1	Variant (**vbDate** format)	Read/write
TimeLast	1.1	Variant (**vbDate** format)	Read/write
Type	1.1	String	Read/write
Unread	1.1	Boolean	Read/write

Methods

Name	Available in version	Parameters
IsSameAs	1.1	*object* as **Object**

Remarks

A MessageFilter object with no criteria is created by default for every Messages collection. This means that initially the filter's properties are unset and its child Fields collection is empty. You specify the filter by setting values for its properties, and by adding fields to its Fields collection and setting a value for each added field.

The filter is invoked when the Messages collection is traversed with the **Get** methods. Each field participates in a MAPI search restriction comparing the field's **Value** property against the value of the Message property specified by the field's **ID** property.

For fields of data type other than **String**, the MAPI search restriction type is RES_PROPERTY with relational operator RELOP_EQ. For fields of data type **String**, the restriction type is RES_CONTENT with fuzzy level options FL_SUBSTRING, FL_IGNORECASE, and FL_LOOSE.

The results of the individual restrictions are normally **AND**ed together to form the final filter value. You can change this by setting the **Or** property, which causes all the results to be **OR**ed instead of **AND**ed. You can also set the **Not** property to specify that the result of each individual restriction is to be negated before being **AND**ed or **OR**ed into the final filter value.

The MessageFilter object is persistent within its parent Messages collection. It is not deleted even when it is released, and it remains attached to the Messages collection until the collection's **Filter** property is set to **Nothing** or the collection is itself released.

Conversation Property (MessageFilter Object)

The **Conversation** property sets filtering on a message's conversation topic. Read/write.

Syntax

objMessageFilter.**Conversation**

Data Type

String

Remarks

The **Conversation** property specifies that the message filter should pass only messages whose conversation topic exactly matches the value of **Conversation**. That is, *objMessageFilter*.**Conversation** sets filtering on *objMessage*.**ConversationTopic**.

A conversation is a group of related messages. The Message object's **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.

For more information on conversations, see Working With Conversations.

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

Fields Property (MessageFilter Object)

The **Fields** property returns a single Field object or a Fields collection object. Read-only.

Syntax

objMessageFilter.**Fields**

objMessageFilter.**Fields(***index***)**

objMessageFilter.**Fields(***proptag***)**

objMessageFilter.**Fields(***name***)**

index
Short integer (less than or equal to 65535). Specifies the index within the Fields collection.

proptag
Long integer (greater than or equal to 65536). Specifies the property tag value for the MAPI property to be retrieved.

name
String. Specifies the name of the custom MAPI property.

Data Type

Object

Remarks

The **Fields** property returns one or more of the fields associated with a MessageFilter object. Each field typically corresponds to a MAPI property, and together the fields that have been added to the collection specify the filter.

The **Fields** property provides a generic access mechanism that allows Visual Basic and Visual C++ programmers to retrieve the value of a MAPI property using either its name or its MAPI property tag. For access with the property tag, use *objMessageFilter*.**Fields**(*proptag*), where *proptag* is the 32-bit MAPI property tag associated with the property, such as **mapiPR_MESSAGE_CLASS**. To access a named property, use *objMessageFilter*.**Fields**(*name*), where *name* is a string that represents the custom property name.

Importance Property (MessageFilter Object)

The **Importance** property sets filtering on a message's importance to **mapiNormal** (the default), **mapiLow**, or **mapiHigh**. Read/write.

Syntax

objMessageFilter.**Importance**

Data Type

Long

Remarks

The following values are defined:

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

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

IsSameAs Method (MessageFilter Object)

The **IsSameAs** method returns **True** if the MessageFilter object is the same as the MessageFilter object being compared against.

Syntax

objMessageFilter.**IsSameAs(***objMsgFilter2***)**

Parameters

objMessageFilter
Required. This MessageFilter object.

objMsgFilter2
Required. The MessageFilter object being compared against.

Remarks

Two MessageFilter objects are considered to be the same if and only if they are instantiations of the same physical (persistent) object. Two objects with the same value are still considered different if they do not instantiate the same physical object, for example if one is a copy of the other. In such a case **IsSameAs** returns **False**.

The **IsSameAs** method ultimately calls one of the MAPI **CompareEntryIDs** methods to determine if two objects are the same. A generic comparison of any two object identifiers is also available with the Session object's **CompareIDs** method.

Not Property (MessageFilter Object)

The **Not** property specifies that all restriction values are to be negated before being **AND**ed or **OR**ed to specify the message filter. Read/write.

Syntax

objMessageFilter.**Not**

Data Type

Boolean

Remarks

If the **Not** property is **False**, the restriction values are treated normally. If it is **True**, each value is toggled (between **True** and **False**) before being used.

Or Property (MessageFilter Object)

The **Or** property specifies that the restriction values are to be **OR**ed instead of **AND**ed to specify the message filter. Read/write.

Syntax

objMessageFilter.**Or**

Data Type

Boolean

Remarks

If the **Or** property is **False**, all the restriction values are **AND**ed together. If it is **True**, the values are **OR**ed together.

Recipients Property (MessageFilter Object)

The **Recipients** property sets filtering on whether a message's recipients include a particular Recipient object. Read/write.

Syntax

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

objOneRecip
Object. A single Recipient object.

objMessageFilter
Object. This MessageFilter object.

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

Data Type

Object

Remarks

Although the **Recipients** property nominally deals with a Recipients collection, only the first member participates in message filtering. You should not maintain more than one Recipient object in the collection.

The **Recipients** property specifies that the message filter should pass only messages with a recipient corresponding to the **Recipients** property. The comparison uses the **Name** property of both Recipient objects. The filter passes the message if the **Name** property of any of its recipients contains the filter recipient's **Name** property as a substring.

Example

This code fragment copies the first valid recipient from an original message to a message filter in order to restrict the Messages collection to messages containing that recipient:

```
Dim objOneRecip as Object
' assume objMessage and objMessageFilter are valid
objMessageFilter.Recipients.Delete ' remove any filter recipients
For i
    strRecipName = objMessage.Recipients.Item(1).Name
    If strRecipName <> "" Then
        Set objOneRecip = objMessageFilter.Recipients.Add
        If objOneRecip Is Nothing Then
            MsgBox "unable to create recipient in message filter"
            Exit Function
        End If
        objOneRecip.Name = strRecipName
        objOneRecip.Address = objMessage.Recipients.Item(i).Address
        objOneRecip.Type = objMessage.Recipients.Item(i).Type
    End If
Next i
```

Sender Property (MessageFilter Object)

The **Sender** property sets filtering on a message's sender to an AddressEntry object. Read/write.

Syntax

Set *objAddrEntry* **=** *objMessageFilter*.**Sender**

objAddrEntry
Object. The AddressEntry object that represents the messaging user that sent the message.

objMessageFilter
Object. The MessageFilter object.

Data Type

Object

Remarks

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

Sent Property (MessageFilter Object)

The **Sent** property sets filtering on whether or not a message was sent through the MAPI system. Read/write.

Syntax

objMessageFilter.**Sent**

Data Type

Boolean

Remarks

A message's **Sent** property is **True** if it was sent through the MAPI system and **False** if it was posted or saved.

In general, there are three different kinds of messages: *sent*, *posted*, and *saved*. Sent messages are traditional e-mail messages sent to a recipient or public folder. Posted messages are created in a public folder. Saved messages are created and saved without either sending or posting. For more information, see the Message object's **Sent** property.

Size Property (MessageFilter Object)

The **Size** property sets filtering on a message's approximate total size in bytes. Read/write.

Syntax

objMessageFilter.**Size**

Data Type

Long

Remarks

The **Size** property specifies that the message filter should pass only messages with approximate total size greater than the value of **Size**.

The **Size** property represents the sum of all the message's MAPI properties, including the **Subject**, **Text**, **Attachments**, and **Recipients**.

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

Subject Property (MessageFilter Object)

The **Subject** property sets filtering on a message's subject. Read/write.

Syntax

objMessageFilter.**Subject**

Data Type

String

Remarks

The **Subject** property specifies that the message filter should pass only messages having a **Subject** that contains the string in this **Subject** property as a substring.

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

The **Subject** property is the default property of a MessageFilter object, meaning that *objMessageFilter* is syntactically equivalent to *objMessageFilter*.**Subject** in Visual Basic code.

Text Property (MessageFilter Object)

The **Text** property sets filtering on a message's main content. Read/write.

Syntax

objMessageFilter.**Text**

Data Type

String

Remarks

The **Text** property specifies that the message filter should pass only messages having a **Text** that contains the string in this **Text** property as a substring.

Note that the **Text** property is a plain text representation of the main portion of the message's content, and does not support formatted text.

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

TimeFirst Property (MessageFilter Object)

The **TimeFirst** property sets filtering on whether a message was received at or since the specified date and time. Read/write.

Syntax

objMessageFilter.**TimeFirst**

Data Type

Variant (**vbDate** format)

Remarks

If the **TimeFirst** property is not set, the message filter passes all messages received at or before the date and time in the **TimeLast** property. If neither property is set, the filter passes messages regardless of their date and time of reception.

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

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

TimeLast Property (MessageFilter Object)

The **TimeLast** property sets filtering on whether a message was received at or before the specified date and time. Read/write.

Syntax

objMessageFilter.**TimeLast**

Data Type

Variant (**vbDate** format)

Remarks

If the **TimeLast** property is not set, the message filter passes all messages received at or since the date and time in the **TimeFirst** property. If neither property is set, the filter passes messages regardless of their date and time of reception.

For more information and an example using the **TimeLast** property, see Filtering Messages in a Folder.

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

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

Type Property (MessageFilter Object)

The **Type** property sets filtering on a message's MAPI message class. Read/write.

Syntax

objMessageFilter.**Type**

Data Type

String

Remarks

The **Type** property specifies that the message filter should pass only messages with a **Type** exactly matching a particular MAPI message class. By default, the OLE Messaging Library sets the **Type** value of new messages to the MAPI message class IPM.Note.

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

Unread Property (MessageFilter Object)

The **Unread** property sets filtering on whether or not a message has been read. Read/write.

Syntax

objMessageFilter.**Unread**

Data Type

Boolean

Remarks

A message's **Unread** property is **True** if it has not been read by the current user.

For more information and an example using the **Unread** property, see Filtering Messages in a Folder.

The **Unread** property corresponds to the MSGFLAG_READ flag not being set in the MAPI property PR_MESSAGE_FLAGS.

Messages Collection Object

The Messages collection object contains one or more Message objects.

Quick Info

Specified in header file:	MDISP.ODL
First available in:	OLE Messaging Library version 1.0.a
Parent objects:	Folder (Inbox or Outbox)
Child objects:	Message MessageFilter
Default property:	Item

A Messages collection is considered a *large collection*, which means that the **Count** and **Item** properties have limited validity, and your best option is to use a Message object identifier value or the **Get** methods to access an individual Message object within the collection.

Properties

Name	Available in version	Type	Access
Application	1.0.a	String	Read-only
Class	1.0.a	Long	Read-only
Count	1.1	Long	Read-only
Filter	1.1	MessageFilter object	Read/write
Item	1.1	Message object	Read-only
Parent	1.0.a	Inbox or Outbox folder object	Read-only
Session	1.0.a	Session object	Read-only

Methods

Name	Available in version	Parameters
Add	1.0.a	(optional) *subject* as **String**, (optional) *text* as **String**, (optional) *type* as **String**, (optional) *importance* as **Long**
Delete	1.0.a	(none)
GetFirst	1.0.a	(optional) *filter* as **String**
GetLast	1.0.a	(optional) *filter* as **String**
GetNext	1.0.a	(none)
GetPrevious	1.0.a	(none)
Sort	1.0.a	(optional) *SortOrder* as **Long**, (optional) *PropTag* as **Long**, (optional) *PropID* as **String**

Remarks

Large collections, such as the Messages collection, cannot always maintain an accurate count of the number of objects in the collection. It is strongly recommended that you use the **GetFirst**, **GetNext**, **GetLast**, and **GetPrevious** methods to access individual items in the collection. You can access one

specific message by using the Session object's **GetMessage** method, and you can access all the items in the collection with the Visual Basic **For Each** construction.

The order that items are returned by **GetFirst**, **GetNext**, **GetLast**, and **GetPrevious** depends on whether the messages are sorted or not. The Message objects within a collection can be sorted on a MAPI property of your choice, either ascending or descending, using the **Sort** method. When the items are not sorted, you should not rely on these methods to return the items in any specified order. The best programming approach to use with unsorted collections is to assume that the access functions are able to access all items within the collection, but that the order of the objects is not defined.

A message and most of its attachments, fields, properties, and recipients are read from the message store when the application first accesses the Message object. For performance reasons, attachment data and field values greater than 1,000 bytes are read from the store only when the application explicitly accesses the Attachment or Field objects. All other properties of the Attachment and Field objects are read when the parent message is read.

See Also

Object Collections

Add Method (Messages Collection)

The **Add** method creates and returns a new Message object in the Messages collection.

Syntax

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

Parameters

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

objMsgColl
Required. The Messages collection object.

subject
Optional. String. The subject of the message. When this parameter is not supplied, the default value is an empty string.

text
Optional. String. The body text of the message. When this parameter is not supplied, the default value is an empty string.

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

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

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

Remarks

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.

The user must have permission to **Add** or **Delete** a Message object. Most users have this permission in their mailbox and their Personal Folders.

The new Message object is saved in the MAPI system when you call its **Update** method.

Example

This code fragment 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
```

Count Property (Messages Collection)

The **Count** property returns the number of Message objects in the collection, or a very large number if the exact count is not available. Read-only.

Syntax

objMsgColl.**Count**

Data Type

Long

Remarks

The **Count** property is useful for determining whether a Messages collection is empty or not.

A large collection cannot always maintain an accurate count of its members, and the **Count** property cannot be used as the collection's size when it has a very large value such as **mapiMaxCount**. Programmers needing to access individual objects in a large collection are strongly advised to use the Visual Basic **For Each** statement or the **Get** methods.

The recommended procedures for traversing a large collection are, in decreasing order of preference:

1. Global selection, such as the Visual Basic **For Each** statement
2. The **Get** methods, particularly **GetFirst** and **GetNext**
1. An indexed loop, such as the Visual Basic **For ... Next** construction

If the message store provider cannot supply the precise number of Message objects, the OLE Messaging Library returns a very large number for the **Count** property. On 32-bit platforms, this value is **mapiMaxCount**, which equals 2^31 - 1, or 2147483647. On other platforms, **mapiMaxCount** is not defined, and the OLE Messaging Library returns -1. A program on such a platform must ensure that -1 does not prematurely terminate any iteration based on the **Count** property.

Programmers using an indexed loop terminating on the **Count** property must also check each returned object for a value of **Nothing**. The loop must proceed forward from the beginning of the collection, and the index must have initial and increment values of 1. Results are undefined for any other procedure.

The use of the **Item** property in conjunction with the **Count** property in a large collection can be seen in the following example.

Example

This code fragment searches for a Message object with subject "Bonus":

```
Dim i As Integer ' loop index / object counter
Dim collMessages as Object ' Messages collection; assume already given
If collMessages Is Nothing Then
    ' MsgBox "Messages collection object is invalid"
    ' Exit
End If
' see if collection is empty
If 0 = collMessages.Count Then
    ' MsgBox "No messages in collection"
    ' Exit
End If
' look for message about "Bonus" in collection
For i = 1 To collMessages.Count Step 1
    If collMessages.Item(i) Is Nothing Then
    ' MsgBox "No such message found in collection"
```

```
    ' Exit ' no more messages in collection
    End If
    If collMessages.Item(i).Subject = “Bonus” Then
    ' MsgBox "Desired message is at index " & i
    ' Exit
    End If
Next i
```

Delete Method (Messages Collection)

The **Delete** method deletes all the messages in the Messages collection.

Syntax

objMsgColl.**Delete()**

Parameters

objMsgColl

Required. The Messages collection object.

Remarks

The **Delete** method moves all the Message objects in the collection to the Deleted Items folder, if the user has enabled this option. If the option is not enabled, or if the Messages collection is already in the Deleted Items folder, the **Delete** method permanently deletes the messages, and they cannot be recovered.

Deleted messages are invalidated but not removed from memory. The programmer should set the invalidated objects to **Nothing** to remove them from memory, or reassign them to other messages. Attempted access to a deleted object results in a return of **mapiE_INVALID_OBJECT**.

Be cautious using the **Delete** method with a collection, because it deletes all the collection's member objects. To delete only one Message object, use the **Delete** method specific to that object.

The **Delete** method on a large collection takes effect immediately and is permanent. A deleted member cannot be recovered. However, the collection itself it still valid, and you can **Add** new members to it.

For example, this code fragment deletes all of the messages in a folder:

```
objFolder.Messages.Delete
```

Filter Property (Messages Collection)

The **Filter** property returns a MessageFilter object for the Messages collection. Read/write.

Syntax

objMsgColl.**Filter**

Data Type

Object

Remarks

A MessageFilter object with no criteria is created by default for every Messages collection. When you specify criteria by setting properties in the filter's Fields collection, the filter restricts any subsequent search on the Messages collection. For more information, see Filtering Messages in a Folder and the MessageFilter Object.

GetFirst Method (Messages Collection)

The **GetFirst** method returns the first Message object in the Messages collection. It 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. String. Specifies the message class of the object, such as the default value, IPM.Note. Corresponds to the **Type** property of the Message object.

Remarks

If the *filter* parameter is set, the **GetFirst** method returns the first message in the collection with a **Type** property matching the value of *filter*.

The **Get** methods are similar to the **Find** and **Move** methods used with Microsoft Access, but they use a different syntax.

GetLast Method (Messages Collection)

The **GetLast** method returns the last Message object in the Messages collection. It 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. String. Specifies the message class of the object, such as the default value, IPM.Note. Corresponds to the **Type** property of the Message object.

Remarks

If the *filter* parameter is set, the **GetFirst** method returns the last message in the collection with a **Type** property matching the value of *filter*.

The **Get** methods are similar to the **Find** and **Move** methods used with Microsoft Access, but they use a different syntax.

GetNext Method (Messages Collection)

The **GetNext** method returns the next Message object in the Messages collection. It returns **Nothing** if no next object exists, for example if already positioned at the end of the collection.

Syntax

Set *objMessage* **=** *objMsgColl*.**GetNext( )**

Parameters

objMessage

On successful return, represents the next Message object in the collection.

objMsgColl

Required. The Messages collection object.

Remarks

The **Get** methods are similar to the **Find** and **Move** methods used with Microsoft Access, but they use a different syntax.

GetPrevious Method (Messages Collection)

The **GetPrevious** method returns the previous Message object in the Messages collection. It returns **Nothing** if no previous object exists, for example if already positioned at the beginning of the collection.

Syntax

Set *objMessage* **=** *objMsgColl*.**GetPrevious( )**

Parameters

objMessage

On successful return, represents the previous Message object in the collection.

objMsgColl

Required. The Messages collection object.

Remarks

The **Get** methods are similar to the **Find** and **Move** methods used with Microsoft Access, but they use a different syntax.

Item Property (Messages Collection)

The **Item** property returns a single Message object from the Messages collection. Read-only.

Syntax

objMsgColl.**Item(***index***)**

objMsgColl.**Item(***prefix***)**

index
A long integer ranging from 1 to the size of the Messages collection.

prefix
A string representing a prefix substring of a Message object's **Subject** property.

Data Type

Object

Remarks

The **Item** property is useful for satisfying syntax requirements when obtaining a member of a Messages collection.

A large collection cannot support true integer indexing, and the **Item(***index***)** syntax cannot be used for arbitrary selection of members of the collection. Programmers needing to access individual objects in a large collection are strongly advised to use the Visual Basic **For Each** statement or the **Get** methods, particularly **GetFirst** and **GetNext**

The **Item(***index***)** syntax is provided solely as a placeholder in an indexed loop, such as the **For ... Next** construction in Visual Basic. Such a loop must proceed forward from the beginning of the collection, and the index must have initial and increment values of 1. Results are undefined for any other procedure.

For more information on using the **Count** and **Item** properties in a large collection, see the example in the **Count** property.

The **Item(***prefix***)** syntax returns the first Message object whose **Subject** property begins with the string specified by *prefix*.

The **Item** property is the default property of a Messages collection, meaning that *objMsgColl***(***index***)** is syntactically equivalent to *objMsgColl*.**Item(***index***)** in Visual Basic code.

Sort Method (Messages Collection)

The **Sort** method sorts the messages in the collection on the specified property according to the specified sort order.

Syntax

objMsgColl.**Sort([***SortOrder, PropTag***])**

objMsgColl.**Sort([***SortOrder, PropID***])**

Parameters

objMsgColl

Required. The Messages collection object.

SortOrder

Optional. Long. The specified sort order, one of the following values:

Value	Numeric value	Description
mapiNone	0	No sort
mapiAscending	1	Ascending sort (default)
mapiDescending	2	Descending sort

PropTag

Optional. Long. The property tag value for the MAPI property to be used for the sort. *PropTag* is the 32-bit MAPI property tag associated with the property, such as **mapiPR_MESSAGE_CLASS**.

PropID

Optional. String. The custom property name of a MAPI named property.

Remarks

Both parameters are optional. If *SortOrder* is not specified, ascending order is used. If neither *PropTag* nor *PropID* is specified, the property used in the previous call to **Sort** is used again. If **Sort** has never been called on this collection during this session, the MAPI property **mapiPR_MESSAGE_DELIVERY_TIME** is used for the sort.

Recipient Object

The Recipient object represents a recipient of a message.

Quick Info

Specified in header file:	MDISP.ODL
First available in:	OLE Messaging Library version 1.0**.a**
Parent objects:	Recipients collection
Child objects:	AddressEntry
Default property:	Name

Properties

Name	Available in version	Type	Access
Address	1.0.a	String	Read/write
AddressEntry	1.0.a	AddressEntry object	Read/write
Application	1.0.a	String	Read-only
Class	1.0.a	Long	Read-only
DisplayType	1.0.a	Long	Read-only
ID	1.1	String	Read/write
Index	1.0.a	Long	Read-only
Name	1.0.a	String	Read/write
Parent	1.0.a	Recipients collection object	Read-only
Session	1.0.a	Session object	Read-only
Type	1.0.a	Long	Read/write

Methods

Name	Available in version	Parameters
Delete	1.0.a	(none)
IsSameAs	1.1	*object* as **Object**
Resolve	1.0.a	(optional) *showDialog* as **Boolean**

Address Property (Recipient Object)

The **Address** property specifies the *full address* for the recipient. Read/write.

Syntax

objRecipient.**Address**

Data Type

String

Remarks

Sets the value of the Recipient object's **Address** property to specify a custom address. The Recipient **Address** uses the following syntax:

AddressType:*AddressValue*

where *AddressType* 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.

The OLE Messaging Library sets the value of the Recipient object's **Address** property for you when you supply the **Name** property and call the recipient's **Resolve** method.

The **Address** property corresponds to the MAPI properties PR_ADDRTYPE and PR_EMAIL_ADDRESS.

Example

```
' from the sample function Util_CompareAddressParts
' assume valid Recipient object
    Set objAddrEntry = objOneRecip.AddressEntry
    strMsg = "Recipient full address = " & objOneRecip.Address
    strMsg = strMsg & "; AddressEntry type = " & objAddrEntry.Type
    strMsg = strMsg & "; AddressEntry address = " & _
                                                objAddrEntry.Address
    MsgBox strMsg ' compare address components
```

AddressEntry Property (Recipient Object)

The **AddressEntry** property contains the AddressEntry object representing the recipient. Read/write.

Syntax

objRecipient.**AddressEntry**

Data Type

Object (AddressEntry object)

Remarks

For a complete description of the relationship between the AddressEntry object and the Recipient object, see Using Addresses.

Accessing the **AddressEntry** property forces resolution of an unresolved recipient name. If the name cannot be resolved, the OLE Messaging Library reports an error. For example, when the recipient contains an empty string, the resolve operation returns **mapiE_AMBIGUOUS_RECIP**.

Example

This code fragment compares the **Address** property of the Recipient object with the **Address** and **Type** properties of its child AddressEntry object, accessible through the recipient's **AddressEntry** property, to demonstrate the relationships between these properties.

```
' 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 display names
    strMsg = "Recipient name = " & objOneRecip.Name
    strMsg = strMsg & "; AddressEntry name = " & objAddrEntry.Name
' Note - the Type properties are NOT the same:
'            AddressEntry.Type is the address type, such as SMTP
'            Recipient.Type is the recipient type, such as To: or Cc:
```

Delete Method (Recipient Object)

The **Delete** method deletes the Recipient object.

Syntax

objRecipient.**Delete()**

Parameters

objRecipient
Required. The Recipient object.

Remarks

The Recipient object is invalidated in memory, but the change is not permanent until you use the **Update**, **Send**, or **Delete** method on the Message object to which this recipient belongs.

The **Delete** operation invalidates the Recipient object but does not remove it from memory. The programmer should set the invalidated object to **Nothing** to remove it from memory, or reassign it to another recipient. Attempted access to a deleted object results in a return of **mapiE_INVALID_OBJECT**.

The immediate parent of this Recipient object is a Recipients collection, which is a child of the message. You can delete all the message's recipients by calling the collection's **Delete** method.

DisplayType Property (Recipient Object)

The **DisplayType** property identifies the nature of the recipient. This property enables special processing based on the type, such as displaying an icon associated with that type. Read-only.

Syntax

objRecipient.**DisplayType**

Parameters

objRecipient
Required. The Recipient object.

Data Type

Long

Remarks

You can use the display type to sort or filter recipients.

The following values are defined:

DisplayType value	Description
mapiUser	Local user
mapiDistList	Distribution list
mapiForum	Public folder
mapiAgent	Agent
mapiOrganization	Organization
mapiPrivateDistList	Private distribution list
mapiRemoteUser	Remote user

ID Property (Recipient Object)

The **ID** property returns the unique identifier of the Recipient object as a string. Read-write.

Syntax

objRecipient.**ID**

Data Type

String

Remarks

MAPI assigns a permanent, unique identifier when an object is created. This identifier does not change from one MAPI session to another, nor from one messaging domain to another.

The **ID** property corresponds to the MAPI property PR_ENTRYID, converted to a string of hexadecimal characters.

Index Property (Recipient Object)

The **Index** property returns the index number of the Recipient object within the Recipients collection. Read-only.

Syntax

objRecipient.**Index**

Data Type

Long

Remarks

The **Index** property indicates this object's position within the parent Recipients collection. It can later be used to reselect this attachment with the collection's **Item** property.

An index value should not be considered a static value that remains constant for the duration of a session. It can be affected when other recipients are added and deleted. The index value is changed following an update to the Message object to which the Recipients collection belongs.

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
```

IsSameAs Method (Recipient Object)

The **IsSameAs** method returns **True** if the Recipient object is the same as the Recipient object being compared against.

Syntax

objRecipient.**IsSameAs(***objRecip2***)**

Parameters

objRecipient
Required. This Recipient object.

objRecip2
Required. The Recipient object being compared against.

Remarks

Two Recipient objects are considered to be the same if and only if they are instantiations of the same physical (persistent) object. Two objects with the same value are still considered different if they do not instantiate the same physical object, for example if one is a copy of the other. In such a case **IsSameAs** returns **False**.

The **IsSameAs** method ultimately calls one of the MAPI **CompareEntryIDs** methods to determine if two objects are the same. A generic comparison of any two object identifiers is also available with the Session object's **CompareIDs** method.

Name Property (Recipient Object)

The **Name** property returns or sets the name of the Recipient object as a string. Read/write.

Syntax

objRecipient.**Name**

Data Type

String

Remarks

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

The **Name** property is the default property of a Recipient object, meaning that *objRecipient* is syntactically equivalent to *objRecipient*.**Name** in Visual Basic code.

Example

```
' from the sample function Util_CompareFullAddressParts()
Dim strMsg As String
    Set objAddrEntry = objOneRecip.AddressEntry
' 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)
```

Resolve Method (Recipient Object)

The **Resolve** method resolves a recipient's address information into a full messaging address.

Syntax

objRecipient.**Resolve([** *showDialog* **])**

Parameters

objRecipient
Required. The Recipient object.

showDialog
Optional. Boolean. If **True** (the default value), displays a modal dialog box to prompt the user to resolve ambiguous names.

Remarks

The **Resolve** method operates when the **AddressEntry** property is set to **Nothing**. Its operation depends on whether you supply the Recipient's **Name** or **Address** property.

When you supply the **Name** property, **Resolve** looks it up in the address book. When a recipient is resolved, the Recipient object's **Address** property is set to the full address and its **AddressEntry** property is set to the child AddressEntry object that represents a copy of information in the address book.

Note that the **Resolve** method does not validate the Recipient object's **Type** property.

When you specify a custom address by supplying only the Recipient object'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 box.

The **Resolve** method uses the address list specified in the profile, such as the global address list (GAL) or the personal address book (PAB).

The Recipients collection's **Resolved** property is set to **True** when every recipient in the collection has its address resolved.

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

See Also

Resolved Property (Recipients Collection)

Type Property (Recipient Object)

The **Type** property specifies the type of the Recipient object, either To, Cc, or Bcc. Read/write.

Syntax

objRecipient.**Type**

Data Type

Long

Remarks

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 (default).
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 contains one or more Recipient objects and specifies the recipients of a message.

Quick Info

Specified in header file:	MDISP.ODL
First available in:	OLE Messaging Library version 1.0.a
Parent objects:	Message
Child objects:	Recipient
Default property:	Item

A Recipients collection is considered a *small collection*, which means that it supports count and index values that let you access an individual Recipient object through the **Item** property. The Recipients collection supports the Visual Basic **For Each** statement.

Properties

Name	Available in version	Type	Access
Application	1.0.a	String	Read-only
Class	1.0.a	Long	Read-only
Count	1.0.a	Long	Read-only
Item	1.0.a	Recipient object	Read-only
Parent	1.0.a	Message object	Read-only
Resolved	1.0.a	Boolean	Read-only
Session	1.0.a	Session object	Read-only

Methods

Name	Available in version	Parameters
Add	1.0.a	(optional) *name* as **String**, (optional) *address* as **String**, (optional) *type* as **Long**, (optional) *entryID* as **String**
Delete	1.0.a	(none)
Resolve	1.0.a	(optional) *showDialog* as **Boolean**

See Also

Object Collections

Add Method (Recipients Collection)

The **Add** method creates and returns 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. When this parameter is not present, the new Recipient object's **Name** property is set to an empty string.

address

Optional. String. The messaging address of the recipient. When this parameter is not present, the new Recipient object's **Address** property is set to an empty string.

type

Optional. Long. The type of recipient; the initial value for the new Recipient object's **Type** property. The following values are valid:

Recipient type	Value	Description
mapiTo	1	The recipient is on the To line (default).
mapiCc	2	The recipient is on the Cc line.
mapiBcc	3	The recipient is on the Bcc line.

When this parameter is not present, the object is given the default value **mapiTo**.

entryID

Optional. String. The unique identifier of a valid AddressEntry object for this recipient. No default value is supplied for the *entryID* parameter. When it is present, the other parameters are not used. When it is not present, the method uses the *name*, *address*, and *type* parameters to determine the recipient.

Remarks

The *name*, *address*, and *type* parameters correspond to the Recipient object's **Name**, **Address**, and **Type** properties, respectively. The *entryID* parameter corresponds to an AddressEntry object's **ID** property. When the *entryID* parameter is present, the other parameters are not used.

When no parameters are present, an empty Recipient object is created.

Call the **Resolve** method after you add a recipient. After the recipient is resolved, you can access the child AddressEntry object through the Recipient object's **AddressEntry** property.

The **Index** property of the new Recipient object equals the new **Count** property of the Recipients collection.

The recipient is saved in the MAPI system when you **Update** or **Send** the parent Message object.

Example

This code fragment 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 existing AddressEntry using ID"
        Exit Function
    End If

    objNewMessage.Text = "expect 3 different recipients"
    MsgBox ("count = " & objNewMessage.Recipients.Count)
```

Count Property (Recipients Collection)

The **Count** property returns the number of Recipient objects in the collection. Read-only.

Syntax

objRecipColl.**Count**

Data Type

Long

Example

This code fragment uses the **Count** property as a loop terminator to copy all Recipient objects from one Recipients collection to another. It shows the **Count** and **Item** properties working together.

```
' 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
        objOneRecip.Address = objOneMsg.Recipients.Item(i).Address
        objOneRecip.Type = objOneMsg.Recipients.Item(i).Type
    End If
Next i
```

Delete Method (Recipients Collection)

The **Delete** method deletes all the recipients in the Recipients collection.

Syntax

objRecipColl.**Delete()**

Parameters

objRecipColl
Required. The Recipients collection object.

Remarks

The **Delete** operation invalidates all the Recipient objects in the collection but does not remove them from memory. The programmer should set the invalidated objects to **Nothing** to remove them from memory, or reassign them to other recipients. Attempted access to a deleted object results in a return of **mapiE_INVALID_OBJECT**.

Be cautious using the **Delete** method with a collection, because it deletes all the collection's member objects. To delete only one Recipient object, use the **Delete** method specific to that object.

The effect of the **Delete** method is not permanent until you use the **Update**, **Send**, or **Delete** method on the parent Message object containing the Recipients collection. Any member once permanently deleted cannot be recovered. However, the collection itself it still valid, and you can **Add** new members to it.

Item Property (Recipients Collection)

The **Item** property returns a single Recipient object from the Recipients collection. Read-only.

Syntax

objRecipColl.**Item(***index***)**

index

A long integer that ranges from 1 to *objRecipColl*.**Count**, or a string that specifies the name of the object.

Data Type

Object

Remarks

The **Item** property works like an accessor property for small collections.

The **Item** property is the default property of a Recipients collection, meaning that *objRecipColl***(***index***)** is syntactically equivalent to *objRecipColl*.**Item(***index***)** in Visual Basic code.

Example

This code fragment shows the **Count** and **Item** properties working together:

```
' list all recipient names in the collection
strRecips = "" ' initialize string
Set objRecipsColl = objOriginalMsg.Recipients
Count = objRecipsColl.Count
For i = 1 To Count Step 1
    Set objOneRecip = objRecipsColl.Item(i) ' or objRecipsColl(i)
    strRecips = strRecips & objOneRecip.Name & "; "
Next i
MsgBox "Message recipients: " & strRecips
```

Resolve Method (Recipients Collection)

The **Resolve** method traverses the Recipients collection to resolve every recipient's address information into a full messaging address.

Syntax

objRecipColl.**Resolve([***showDialog***])**

Parameters

objRecipColl
Required. The Recipients collection object.

showDialog
Optional. Boolean. If **True** (the default value), displays a modal dialog box to prompt the user to resolve ambiguous names.

Remarks

Calling the Recipients collection's **Resolve** method is similar to calling the **Resolve** method for each Recipient object in the collection, except that it also forces an update to the **Count** and **Item** properties and to all Recipient objects in the collection. Any Recipient variable previously set to an object in the collection is invalidated by the collection's **Resolve** method and should be retrieved again from the collection. Note that the individual recipient's **Resolve** method does not invalidate the object.

The **Resolved** property is set to **True** when every recipient in the collection has its address resolved.

The following methods can invoke MAPI dialog boxes: **Delete** and **Details** methods (AddressEntry object), **Options** and **Send** methods (Message object), **Resolve** method (Recipient object), **Resolve** method (Recipients collection), **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 ' resolve and update everything
    End With
```

Resolved Property (Recipients Collection)

The **Resolved** property contains **True** if all of the recipients in the collection have their address information resolved. Read-only.

Syntax

objRecipColl.**Resolved**

Data Type

Boolean

Remarks

A Recipient object is considered resolved when it has a valid AddressEntry object in its **AddressEntry** property.

You should resolve all addresses. Whenever you 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 **Resolved** property is not **True**, use either the collection's **Resolve** method or each individual recipient's **Resolve** method to resolve all the addresses.

When you use existing valid AddressEntry objects, you do not need to explicitly call the **Resolve** method.

Session Object

The Session object contains session-wide settings and options. It also contains properties that return top-level objects, such as **CurrentUser**.

Quick Info

Specified in header file:	MDISP.ODL
First available in:	OLE Messaging Library version 1.0.a
Parent objects:	(none)
Child objects:	AddressLists collection Folder (Inbox or Outbox) InfoStores collection
Default property:	Name

Properties

Name	Available in version	Type	Access
AddressLists	1.1	AddressList object or AddressLists collection object	Read-only
Application	1.0.a	String	Read-only
Class	1.0.a	Long	Read-only
CurrentUser	1.0.a	AddressEntry object	Read-only
Inbox	1.0.a	Folder object	Read-only
InfoStores	1.0.a	InfoStores object	Read-only
MAPIOBJECT	1.0.a	Object	Read/write (Note: Not available to Visual Basic applications.)
Name	1.0.a	String	Read-only
OperatingSystem	1.0.a	String	Read-only
Outbox	1.0.a	Folder object	Read-only
Parent	1.0.a	Object; set to **Nothing**	Read-only
Session	1.0.a	Object; set to **Nothing**	Read-only
Version	1.0.a	String	Read-only

Methods

Name	Available in version	Parameters
AddressBook	1.0.a	(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**
CompareIDs	1.1	*ID1* as **Object**, *ID2* as **Object**
DeliverNow	1.1	(none)
GetAddressEntry	1.0.a	*entryID* as **String**
GetFolder	1.0.a	*folderID* as **String**, *storeID* as **String**
GetInfoStore	1.0.a	*storeID* as **String**
GetMessage	1.0.a	*messageID* as **String**, *storeID* as **String**
Logoff	1.0.a	(none)
Logon	1.0.a	(optional) *profileName* as **String**, (optional) *profilePassword* as **String**, (optional) *showDialog* as **Boolean**, (optional) *newSession* as **Boolean**, (optional) *parentWindow* as **Long**

Remarks

A Session object is considered a top-level object, meaning it can be created directly from a Visual Basic program. The following sample code fragment creates a Session object through early binding:

```
Dim objSession as MAPI.Session
objSession.Logon
```

The following sample code fragment creates a Session object through late binding:

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

After you create a new Session object, use the **Logon** method to initiate a session with MAPI.

AddressBook Method (Session Object)

The **AddressBook** method displays a modal 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 values for the recipient list boxes in the address book. (Note: This initial Recipient collection is ignored in the OLE Messaging Library.)

title

Optional. String. The title or caption of the address book dialog box. The default value is an empty string.

oneAddress

Optional. Boolean. Allows the user to enter or select only one address entry. The default value is **False**.

forceResolution

Optional. Boolean. If **True**, attempts to resolve all names before closing the address book. Prompts the user to resolve any ambiguous names. The default value is **True**.

recipLists

Optional. Long. The number of recipient list boxes to display in the address book dialog box:

recipLists	Action
0	Displays no list boxes. The user can interact with the address book dialog box but no recipients are returned by this method.
1	Displays one list box for **mapiTo** recipients (default).
2	Displays two list boxes for **mapiTo** and **mapiCc** recipients.
3	Displays three list boxes for **mapiTo**, **mapiCc**, and **mapiBcc** recipients.

toLabel

Optional. String. The caption for the button associated with 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 button associated with 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 button associated with 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 zero (the default) specifies that the dialog should be application-modal.

Remarks

The **AddressBook** dialog is always modal, meaning the parent window is disabled while the dialog is active. If the *parentWindow* parameter is set to zero or is not set, all windows belonging to the application are disabled while the dialog is active. If the *parentWindow* parameter is supplied but is not valid, the call returns **mapiE_INVALID_PARAMETER**.

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 label argument string, immediately before the character that serves as the access key. For example, if *toLabel* is “Local &Attendees:”, users can press ALT+A to move the focus to the first recipient list box.

When you use the **AddressBook** method to let the user select recipients for a new message, you must use two different Recipients collection collections. Use the following procedure:

1. Call **AddressBook**, which returns a new Recipients collection.
2. Call **Add** in a Messages collection collection to create a new message.
3. Copy the Recipients collection returned by **AddressBook** to the Recipients property of the new message:

```
Set objNewMessage.Recipients = objRecipients
objNewMessage.Recipients.Resolve ' also updates everything
```

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

Note The initial Recipients collection, as specified in the *recipients* parameter, is not used in the OLE Messaging Library.

Example

The following code fragment displays an address book dialog box labeled “Select Attendees” with three recipient lists:

```
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:=3, _
                  toLabel:="&Very Important People", _ ' on button
                  ccLabel:="&Carbon Recipients", _
                  bccLabel:="&Secret Recipients")
' parameter not used in call: Recipients:=objInitRecipColl
MsgBox "Name of first recipient = " & objRecipColl.Item(1).Name
Exit Function
```

AddressLists Property (Session Object)

The **AddressLists** property returns a single AddressList object or an AddressLists collection object. Read-only.

Syntax

Set *objAddrListsColl* = *objSession.***AddressLists**

Set *objOneAddrList* = *objSession.***AddressLists(***index***)**

objAddrListsColl
Object. An AddressLists collection object.

objSession
Object. The Session object.

objOneAddrList
Object. A single AddressList object.

index
Long. Specifies the number of the address list within the AddressLists collection. Ranges from 1 to the value specified by the AddressLists collection's **Count** property.

Data Type

Object

Remarks

The AddressLists collection represents the root of the MAPI address book hierarchy for the current session. A particular AddressList object represents one of the available address books. The type of access you obtain depends on the access granted to you by each individual address book provider.

CompareIDs Method (Session Object)

The **CompareIDs** method determines whether two OLE Messaging Library objects are the same object.

Syntax

objSession.**CompareIDs(***ID1, ID2***)**

Parameters

objSession
Required. The Session object.

ID1
Required. The identifier of the first object to be compared.

ID2
Required. The identifier of the second object to be compared.

Remarks

The **CompareIDs** method compares the identifiers of two arbitrary OLE Messaging Library objects and returns **True** if they are the same object. Two objects are considered to be the same if and only if they are instantiations of the same physical (persistent) object. Two objects with the same value are still considered different if they do not instantiate the same physical object, for example if one is a copy of the other. In such a case **CompareIDs** returns **False**.

The **CompareIDs** method ultimately calls one of the MAPI **CompareEntryIDs** methods to determine if two objects are the same. Several OLE Messaging Library objects also provide the **IsSameAs** method for a comparison of two identifiers of that particular object type.

CurrentUser Property (Session Object)

The **CurrentUser** property returns the active user as an AddressEntry object. Read-only.

Syntax

objSession.**CurrentUser**

Data Type

Object

Remarks

The **CurrentUser** property returns **Nothing** when no user is logged on.

Example

This code fragment checks for logon, then displays the full messaging address of 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
```

DeliverNow Method (Session Object)

The **DeliverNow** method requests immediate delivery of all undelivered messages submitted in the current session.

Syntax

objSession.**DeliverNow()**

Parameters

objSession
Required. The Session object.

Remarks

The **DeliverNow** method ultimately calls the MAPI spooler's **IMAPIStatus::FlushQueues** method to request that all messages in all inbound and outbound queues be received or delivered immediately. **FlushQueues** is invoked synchronously, and performance degradation is possible during the processing of this request.

GetAddressEntry Method (Session Object)

The **GetAddressEntry** method 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. Specifies the unique identifier of the address entry.

Remarks

For more information, see Using Addresses.

Example

This code fragment 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 to address entry ID")
        Exit Function
    End If
    Set objAddrEntry = objSession.GetAddressEntry(strAddressEntryID)
    MsgBox "Full address = " & objAddrEntry.Type & ":" _
                            & objAddrEntry.Address
```

GetFolder Method (Session Object)

The **GetFolder** method returns a Folder object from a MAPI message store.

Syntax

Set *objFolder* = *objSession*.**GetFolder(***folderID* **[**, *storeID***])**

Parameters

objFolder

On successful return, contains the Folder object with the specified identifier. When the folder does not exist, **GetFolder** returns **Nothing**.

objSession

Required. The Session object.

folderID

Required. String that specifies the unique identifier of the folder. When you provide an empty string, some providers return the root folder.

storeID

Optional. String that specifies the unique identifier of the message store containing the folder. The default value is an empty string, which corresponds to the default message store.

Remarks

The **GetFolder** method allows you to obtain any Folder object for which you know the identifier, that is, the folder's **ID** property.

For some message stores, you can obtain the store's root folder by supplying an empty string as the value for *folderID*. If the message store does not support returning its root folder, the call returns the error value **mapiE_NOT_FOUND**.

Note that the store's root folder differs from the IPM root folder. The store's root folder is the parent of the root folder of the the IPM subtree. The IPM subtree contains all interpersonal messages in a hierarchy of folders. Interpersonal messages are those whose message class starts with IPM, such as IPM.Note.

You can obtain the IPM root folder with the InfoStore object's **RootFolder** property. You can obtain the store's root folder through the IPM root folder's **FolderID** property.

Example

This code fragment uses the **GetFolder** method to obtain a specific folder from a MAPI message store:

```
' from the function Session_GetFolder
' requires a global variable that contains the folder ID
' uses a global variable that contains the message store ID if present
    If strFolderID = "" Then
        MsgBox ("Must first set string variable to folder ID")
        Exit Function
    End If
    If strFolderStoreID = "" Then ' maybe get root folder
        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
Else
    Set objMessages = objFolder.Messages
    MsgBox "Folder set to " & objFolder.Name
End If
```

GetInfoStore Method (Session Object)

The **GetInfoStore** method returns an InfoStore object that can be used to navigate through both public folders and the user's personal folders.

Syntax

Set *objInfoStore* = *objSession*.**GetInfoStore(***storeID***)**

Parameters

objInfoStore

On successful return, contains the InfoStore object with the specified identifier. When the InfoStore object does not exist, **GetInfoStore** returns **Nothing**.

objSession

Required. The Session object.

storeID

Required. String. Specifies the unique identifier of the InfoStore object to retrieve.

Remarks

The **GetInfoStore** method allows you to obtain any message store for which you know the **ID** property. Within the message store you can then obtain any child Folder Object object for which you know the **ID** property.

Example

This code fragment uses the **GetInfoStore** method to obtain a specific message store:

```
' from the function Session_GetInfoStore
' requires a global variable that contains the InfoStore ID
Dim strInfoStoreID as String ' ID as hex string
Dim objInfoStore As Object ' InfoStore object
    If strInfoStoreID = "" Then
        MsgBox ("Must first set string variable to InfoStore ID")
        Exit Function
    End If
    Set objInfoStore = objSession.GetInfoStore( _
                                storeID:=strInfoStoreID)
    ' error handling ...
    MsgBox "InfoStore set to " & objInfoStore.Name
```

GetMessage Method (Session Object)

The **GetMessage** method returns a Message object from a MAPI message store.

Syntax

Set *objMessage* = *objSession*.**GetMessage(***messageID* **[**, *storeID***])**

Parameters

objMessage
On successful return, contains the Message object with the specified identifier. When the specified *messageID* does not exist, **GetMessage** returns **Nothing**.

objSession
Required. The Session object.

messageID
Required. String. Specifies the unique identifier of the message.

storeID
Optional. String. Specifies the unique identifier of the message store. The default value is an empty string, which corresponds to the default message store.

Remarks

The **GetMessage** method allows you to obtain directly any Message Object object for which you know the **ID** property. You do not have to find and open the folder containing the message or the InfoStore containing the folder.

Example

This code fragment displays the subject of a message from a MAPI message store:

```
' fragment from Session_GetMessage
' requires the parameter strMessageID;
' also uses strMessageStoreID if it is defined
If strMessageID = "" Then
    MsgBox ("Must first set string variable to 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
```

Inbox Property (Session Object)

The **Inbox** property returns a Folder object representing the current user's Inbox folder. Read-only.

Syntax

objSession.**Inbox**

Data Type

Object (Folder object)

Remarks

The **Inbox** property returns **Nothing** if the current user does not have an Inbox folder.

In addition to the general ability to navigate through the formal collection and object hierarchy, the OLE Messaging Library supports properties that allow your application to directly access the most common Folder objects:

- The InfoStore object's **RootFolder** property for the IPM subtree root folder
- The Session object's **Inbox** property for the Inbox folder
- The Session object's **Outbox** property for the Outbox folder

Example

This code fragment uses the Session object's **Inbox** property to initialize a Folder Object object:

```
' 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 "Inbox 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
```

InfoStores Property (Session Object)

The **InfoStores** property returns an InfoStores collection available to this session. Read-only.

Syntax

objSession.**InfoStores**

Data Type

Object (InfoStores collection)

Remarks

The **InfoStores** property returns a collection of available message stores. Each InfoStore object in the collection represents an individual message store and provides access to its folder hierarchy.

You can access public folders through the InfoStores collection. The public folders are maintained in their own InfoStore object, which is distinct from the InfoStore object that contains the user's personal messages.

When you know the unique identifier for a particular InfoStore object, you can obtain it directly with the Session object's **GetInfoStore** method.

Example

```
' from the functions Session_InfoStores, InfoStores_FirstItem,
' and InfoStore.Name
Dim objSession as Object         ' Session object
Dim objInfoStoresColl as Object  ' InfoStores collection
Dim objInfoStore as Object       ' InfoStore object
' assume valid Session object
    Set objInfoStoresColl = objSession.InfoStores
    If objInfoStoresColl Is Nothing Then
        MsgBox "Could not set InfoStores collection"
        Exit Function
    End If
    If 0 = objInfoStoresColl.Count Then
        MsgBox "No InfoStores in the collection"
        Exit Function
    End If
    iInfoStoresCollIndex = 1
    Set objInfoStore = objInfoStoresColl.Item(iInfoStoresCollIndex)
    If objInfoStore Is Nothing Then
        MsgBox "Error, cannot get this InfoStore object"
        Exit Function
    Else
        MsgBox "Selected InfoStores " & iInfoStoresCollIndex
    End If
    If "" = objInfoStore.Name Then
        MsgBox "Active InfoStore has no name; ID = " & objInfoStore.Id
    Else
        MsgBox "Active InfoStore has name: " & objInfoStore.Name
    End If
```

Logoff Method (Session Object)

The **Logoff** method logs off from the MAPI system.

Syntax

objSession.**Logoff()**

Parameters

objSession
Required. The Session object.

Example

This code fragment 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)

The **Logon** method logs on to the MAPI system.

Syntax

objSession.**Logon([** *profileName, profilePassword, showDialog, newSession, parentWindow* **])**

Parameters

objSession

Required. The Session object.

profileName

Optional. String. Specifies the user's logon name. To prompt the user to enter a logon name, omit *profileName* and set *showDialog* to **True**. The default value is an empty string.

profilePassword

Optional. String. Specifies the user's logon password. To prompt the user to enter a logon password, omit *profilePassword* and set *showDialog* to **True**. The default value is an empty string.

showDialog

Optional. Boolean. If **True**, displays a logon dialog box. The default value is **True**.

newSession

Optional. Boolean. Determines whether the application opens a new MAPI session or uses the current shared MAPI session (the default). If a shared MAPI session does not exist, *newSession* is ignored and a new session is opened. If a shared MAPI session does exist, this parameter causes the following action:

Value	Action
True	Opens an new MAPI session.
False	Uses the current shared MAPI session (default).

parentWindow

Optional. Long. Specifies the parent window handle for the logon dialog box. A value of **0** (the default) specifies that the dialog should be application-modal. The *parentWindow* parameter is ignored unless *showDialog* is **True**.

Remarks

The user must log on before your application can use most MAPI objects.

The **Logon** dialog is always modal, meaning the parent window is disabled while the dialog is active. If the *parentWindow* parameter is set to **0** or is not set, all windows belonging to the application are disabled while the dialog is active. If the *parentWindow* parameter is supplied but is not valid, the call returns **mapiE_INVALID_PARAMETER**.

The common MAPI dialog boxes automatically handle many of the error cases that can be encountered during logon. When you call **Logon** and do not supply the optional profile name parameter, the **Choose Profile** dialog box appears, asking the user to select a profile. When the *profileName* parameter is supplied but is not valid, common dialog boxes indicate the error and prompt the user to enter a valid name from the **Choose Profile** dialog box. When no profiles are defined, the Profile Wizard takes the user through the creation of a new profile.

If your application calls the **Logon** method after the user has already successfully logged on, the OLE Messaging Library generates the error **mapiE_LOGON_FAILURE**.

For more information, see Starting an OLE Messaging Session.

The following methods can invoke MAPI dialog boxes: **Delete** and **Details** methods (AddressEntry object), **Options** and **Send** methods (Message object), **Resolve** method (Recipient object), **Resolve**

method (Recipients collection), **AddressBook** and **Logon** methods (Session object).

Example

The first code fragment displays a logon dialog box that prompts the user to enter a logon password. The second code fragment supplies the *profileName* parameter and does not display the dialog box:

```
' 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 log on: 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

Logoff Method (Session Object)

MAPIOBJECT Property (Session Object)

The **MAPIOBJECT** property returns an **IUnknown** pointer to the Session object. Not available to Visual Basic applications. Read/write.

Syntax

objSession.**MAPIOBJECT**

Data Type

Variant (**vbDataObject** format)

Remarks

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 Introduction to Automation and How Programmable Objects Work. Also see the Microsoft *OLE Programmer's Reference*.

Name Property (Session Object)

The **Name** property returns the display name of the profile logged on to this session. Read-only.

Syntax

objSession.**Name**

Data Type

String

Remarks

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

The **Name** property is the default property of a Session object, meaning that *objSession* is syntactically equivalent to *objSession*.**Name** in Visual Basic code.

Examples

```
' from the function Session_Name
    If objSession Is Nothing Then
        MsgBox "Must log on first: see Session menu"
        Exit Function
    End If
    MsgBox "Profile name for this session = " & objSession.Name
```

OperatingSystem Property (Session Object)

The **OperatingSystem** property returns the name and version number of the current operating system. Read-only.

Syntax

objSession.**OperatingSystem**

Data Type

String

Remarks

The OLE Messaging Library returns strings in the following formats:

Operating system	String value
Microsoft Windows for Workgroups	Microsoft® Windows™ *N.kk*
Microsoft Windows NT	Microsoft® Windows NT™ *N.kk*

The *N.kk* 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.

The version number returned in the **OperatingSystem** property is not related to the version number returned in the **Version** property.

Example

This code fragment displays the name and version of the operating system:

```
' from the function Session_OperatingSystem
' assume objSession is a valid Session object
    MsgBox "Operating system = " & objSession.OperatingSystem
```

Outbox Property (Session Object)

The **Outbox** property returns a Folder object representing the current user's Outbox folder. Read-only.

Syntax

objSession.**Outbox**

Data Type

Object

Remarks

The **Outbox** property returns **Nothing** if the current user does not have or has not enabled the Outbox folder.

In addition to the general ability to navigate through the formal collection and object hierarchy, the OLE Messaging Library supports properties that allow your application to directly access the most common Folder objects:

- The InfoStore object's **RootFolder** property for the IPM subtree root folder
- The Session object's **Inbox** property for the Inbox folder
- The Session object's **Outbox** property for the Outbox folder

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 "Outbox folder name = " & objFolder.Name
    Set objMessages = objFolder.Messages
```

Version Property (Session Object)

The **Version** property returns the version number of the OLE Messaging Library as a string, for example “1.1”. Read-only.

Syntax

objSession.**Version**

Data Type

String

Remarks

The version number for the OLE Messaging Library is represented by a string in the form *N*.*kk*, where *N* represents a major version number and *kk* represents a minor version number.

The version number returned in the **Version** property is not related to the version number returned in the **OperatingSystem** property.

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 Library version " & _
                                        objSession.Version
```

Error Codes

The desired return value from OLE Messaging Library calls to MAPI is zero, meaning the call was successful and produced the expected results. MAPI can also return either a warning value or an error value to the OLE Messaging Library. A warning means the call was at least partially successful but may have produced an unexpected result or side effect. An error means the call was not successful.

All warning and error return codes are nonzero. Warning values have the high-order bit zero, while error values set it to one. For the convenience of the Visual Basic programmer, the OLE Messaging Library defines type values for all relevant warning and error codes. These are provided here in alphabetic and then in numeric order.

The following table lists the return values from MAPI in alphabetic order:

Warning or error code value	HRESULT (VB4 error value) (hexadecimal)	Low-order word + 1000 (VBA error value) (decimal)
mapiE_ACCOUNT_DISABLED	0x80040124	1292
mapiE_AMBIGUOUS_RECIP	0x80040700	2792
mapiE_BAD_CHARWIDTH	0x80040103	1259
mapiE_BAD_COLUMN	0x80040118	1280
mapiE_BAD_VALUE	0x80040301	1769
mapiE_BUSY	0x8004010B	1267
mapiE_CALL_FAILED	0x80004005	17389
mapiE_CANCEL	0x80040501	2281
mapiE_COLLISION	0x80040604	2540
mapiE_COMPUTED	0x8004011A	1282
mapiE_CORRUPT_DATA	0x8004011B	1283
mapiE_CORRUPT_STORE	0x80040600	2536
mapiE_DECLINE_COPY	0x80040306	1774
mapiE_DISK_ERROR	0x80040116	1278
mapiE_END_OF_SESSION	0x80040200	1512
mapiE_EXTENDED_ERROR	0x80040119	1281
mapiE_FAILONEPROVIDER	0x8004011D	1285
mapiE_FOLDER_CYCLE	0x8004060B	2547
mapiE_HAS_FOLDERS	0x80040609	2545
mapiE_HAS_MESSAGES	0x8004060A	2546
mapiE_INTERFACE_NOT_SUPPORTED	0x80004002	17386
mapiE_INVALID_ACCESS_TIME	0x80040123	1291
mapiE_INVALID_BOOKMARK	0x80040405	2029
mapiE_INVALID_ENTRYID	0x80040107	1263
mapiE_INVALID_OBJECT	0x80040108	1264
mapiE_INVALID_PARAMETER	0x80070057	1087
mapiE_INVALID_TYPE	0x80040302	1770

mapiE_INVALID_WORKSTATION_ACCOUNT	0x80040122	1290
mapiE_LOGON_FAILED	0x80040111	1273
mapiE_MISSING_REQUIRED_COLUMN	0x80040202	1514
mapiE_NETWORK_ERROR	0x80040115	1277
mapiE_NO_ACCESS	0x80070005	1005
mapiE_NO_RECIPIENTS	0x80040607	2543
mapiE_NO_SUPPORT	0x80040102	1258
mapiE_NO_SUPPRESS	0x80040602	2538
mapiE_NON_STANDARD	0x80040606	2542
mapiE_NOT_ENOUGH_DISK	0x8004010D	1269
mapiE_NOT_ENOUGH_MEMORY	0x8007000E	1014
mapiE_NOT_ENOUGH_RESOURCES	0x8004010E	1270
mapiE_NOT_FOUND	0x8004010F	1271
mapiE_NOT_IN_QUEUE	0x80040601	2537
mapiE_NOT_INITIALIZED	0x80040605	2541
mapiE_NOT_ME	0x80040502	2282
mapiE_OBJECT_CHANGED	0x80040109	1265
mapiE_OBJECT_DELETED	0x8004010A	1266
mapiE_PASSWORD_CHANGE_REQUIRED	0x80040120	1288
mapiE_PASSWORD_EXPIRED	0x80040121	1289
mapiE_SESSION_LIMIT	0x80040112	1274
mapiE_STRING_TOO_LONG	0x80040105	1261
mapiE_SUBMITTED	0x80040608	2544
mapiE_TABLE_EMPTY	0x80040402	2026
mapiE_TABLE_TOO_BIG	0x80040403	2027
mapiE_TIMEOUT	0x80040401	2025
mapiE_TOO_BIG	0x80040305	1773
mapiE_TOO_COMPLEX	0x80040117	1279
mapiE_TYPE_NO_SUPPORT	0x80040303	1771
mapiE_UNABLE_TO_ABORT	0x80040114	1276
mapiE_UNABLE_TO_COMPLETE	0x80040400	2024
mapiE_UNCONFIGURED	0x8004011C	1284
mapiE_UNEXPECTED_ID	0x80040307	1775
mapiE_UNEXPECTED_TYPE	0x80040304	1772
mapiE_UNKNOWN_CPID	0x8004011E	1286
mapiE_UNKNOWN_ENTRYID	0x80040201	1513
mapiE_UNKNOWN_FLAGS	0x80040106	1262
mapiE_UNKNOWN_LCID	0x8004011F	1287
mapiE_USER_CANCEL	0x80040113	1275
mapiE_VERSION	0x80040110	1272
mapiE_WAIT	0x80040500	2280
mapiW_APPROX_COUNT	0x00040482	2154
mapiW_CANCEL_MESSAGE	0x00040580	2408
mapiW_ERRORS_RETURNED	0x00040380	1896

mapiW_NO_SERVICE	0x00040203	1515
mapiW_PARTIAL_COMPLETION	0x00040680	2664
mapiW_POSITION_CHANGED	0x00040481	2153

The following table lists the return values from MAPI in numeric order:

HRESULT (VB4 error value) (hexadecimal)	**Low-order word + 1000 (VBA error value) (decimal)**	**Warning or error code value**
0x00040203	1515	**mapiW_NO_SERVICE**
0x00040380	1896	**mapiW_ERRORS_RETURNED**
0x00040481	2153	**mapiW_POSITION_CHANGED**
0x00040482	2154	**mapiW_APPROX_COUNT**
0x00040580	2408	**mapiW_CANCEL_MESSAGE**
0x00040680	2664	**mapiW_PARTIAL_COMPLETION**
0x80004002	17386	**mapiE_INTERFACE_NOT_SUPPORTED**
0x80004005	17389	**mapiE_CALL_FAILED**
0x80040102	1258	**mapiE_NO_SUPPORT**
0x80040103	1259	**mapiE_BAD_CHARWIDTH**
0x80040105	1261	**mapiE_STRING_TOO_LONG**
0x80040106	1262	**mapiE_UNKNOWN_FLAGS**
0x80040107	1263	**mapiE_INVALID_ENTRYID**
0x80040108	1264	**mapiE_INVALID_OBJECT**
0x80040109	1265	**mapiE_OBJECT_CHANGED**
0x8004010A	1266	**mapiE_OBJECT_DELETED**
0x8004010B	1267	**mapiE_BUSY**
0x8004010D	1269	**mapiE_NOT_ENOUGH_DISK**
0x8004010E	1270	**mapiE_NOT_ENOUGH_RESOURCES**
0x8004010F	1271	**mapiE_NOT_FOUND**
0x80040110	1272	**mapiE_VERSION**
0x80040111	1273	**mapiE_LOGON_FAILED**
0x80040112	1274	**mapiE_SESSION_LIMIT**
0x80040113	1275	**mapiE_USER_CANCEL**
0x80040114	1276	**mapiE_UNABLE_TO_ABORT**
0x80040115	1277	**mapiE_NETWORK_ERROR**
0x80040116	1278	**mapiE_DISK_ERROR**
0x80040117	1279	**mapiE_TOO_COMPLEX**
0x80040118	1280	**mapiE_BAD_COLUMN**
0x80040119	1281	**mapiE_EXTENDED_ERROR**
0x8004011A	1282	**mapiE_COMPUTED**
0x8004011B	1283	**mapiE_CORRUPT_DATA**
0x8004011C	1284	**mapiE_UNCONFIGURED**
0x8004011D	1285	**mapiE_FAILONEPROVIDER**

0x8004011E	1286	**mapiE_UNKNOWN_CPID**
0x8004011F	1287	**mapiE_UNKNOWN_LCID**
0x80040120	1288	**mapiE_PASSWORD_CHANGE_REQUIRED**
0x80040121	1289	**mapiE_PASSWORD_EXPIRED**
0x80040122	1290	**mapiE_INVALID_WORKSTATION_ACCOUNT**
0x80040123	1291	**mapiE_INVALID_ACCESS_TIME**
0x80040124	1292	**mapiE_ACCOUNT_DISABLED**
0x80040200	1512	**mapiE_END_OF_SESSION**
0x80040201	1513	**mapiE_UNKNOWN_ENTRYID**
0x80040202	1514	**mapiE_MISSING_REQUIRED_COLUMN**
0x80040301	1769	**mapiE_BAD_VALUE**
0x80040302	1770	**mapiE_INVALID_TYPE**
0x80040303	1771	**mapiE_TYPE_NO_SUPPORT**
0x80040304	1772	**mapiE_UNEXPECTED_TYPE**
0x80040305	1773	**mapiE_TOO_BIG**
0x80040306	1774	**mapiE_DECLINE_COPY**
0x80040307	1775	**mapiE_UNEXPECTED_ID**
0x80040400	2024	**mapiE_UNABLE_TO_COMPLETE**
0x80040401	2025	**mapiE_TIMEOUT**
0x80040402	2026	**mapiE_TABLE_EMPTY**
0x80040403	2027	**mapiE_TABLE_TOO_BIG**
0x80040405	2029	**mapiE_INVALID_BOOKMARK**
0x80040500	2280	**mapiE_WAIT**
0x80040501	2281	**mapiE_CANCEL**
0x80040502	2282	**mapiE_NOT_ME**
0x80040600	2536	**mapiE_CORRUPT_STORE**
0x80040601	2537	**mapiE_NOT_IN_QUEUE**
0x80040602	2538	**mapiE_NO_SUPPRESS**
0x80040604	2540	**mapiE_COLLISION**
0x80040605	2541	**mapiE_NOT_INITIALIZED**
0x80040606	2542	**mapiE_NON_STANDARD**
0x80040607	2543	**mapiE_NO_RECIPIENTS**
0x80040608	2544	**mapiE_SUBMITTED**
0x80040609	2545	**mapiE_HAS_FOLDERS**
0x8004060A	2546	**mapiE_HAS_MESSAGES**
0x8004060B	2547	**mapiE_FOLDER_CYCLE**
0x80040700	2792	**mapiE_AMBIGUOUS_RECIP**
0x80070005	1005	**mapiE_NO_ACCESS**
0x8007000E	1014	**mapiE_NOT_ENOUGH_MEMORY**
0x80070057	1087	**mapiE_INVALID_PARAMETER**

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 appendix provides a very short introduction to the Microsoft Component Object Model, Automation, and the OLE programmability interface **IDispatch**. For complete details, see the *OLE Programmer's Reference*.

You do not need to understand this material 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 (called *methods*). There are no facilities for directly accessing data within an interface, except through the methods.

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 fewer that you don’t.

By convention, interface names start with the letter “I”. The methods are given a specific ordering within the interface. Knowing the order of the methods 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 methods 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 methods in all interfaces are always **QueryInterface** (which developers call “QI”), **AddRef**, and **Release**. These methods 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 file gets loaded and unloaded. Any other methods in the interface are defined by the person who creates the interface. The interface that consists of these three common methods, **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 methods and the data items (called *properties*) 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 one person to define an interface, a second person to implement the interface, and a third person to write a program that calls the 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 methods are present in the interface and what properties are 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 adds 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 method of the interface. **IDispatch** adds a level of indirection. Instead of directly calling the member method of the interface, the program calls **IDispatch::Invoke**, and **IDispatch::Invoke** calls the member method 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, **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 a 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 or C++ programmers would obtain from a header file: the name of the method and the sequence and types of its parameters.

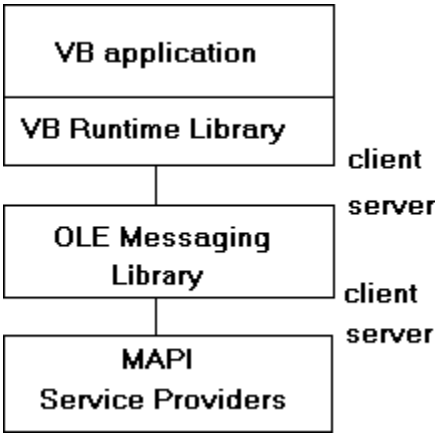
An executable file or DLL that exposes **IDispatch** and its type library is known as an *Automation server*. The OLE Messaging Library is such a server.

The OLE Messaging Library: An Automation Server

So, let's put it all together, from the bottom up, to see how the OLE Messaging Library works.

- Service providers implement COM interfaces—specifically, the MAPI interfaces—as described in the MAPI documentation.
- The OLE Messaging Library implements several objects (Session, Message, and so on) that act as *clients* to these MAPI interfaces. That is, the OLE Messaging Library objects obtain pointers to the MAPI interfaces and call methods.
- The OLE Messaging Library implements **IDispatch** and acts as an 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 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 Visual Basic 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. Visual Basic is a client to the Automation server, the OLE Messaging Library. The OLE Messaging Library, in turn, acts as a client to the MAPI services.



The OLE Messaging Library and MAPI

The OLE Messaging Library calls Microsoft COM and MAPI interfaces for you. The following table describes the MAPI interfaces that the OLE Messaging Library calls when you manipulate an OLE Messaging Library object.

OLE Messaging Library object	COM or MAPI interface called by the OLE Messaging Library
AddressEntry	IABContainer, IMAPIProp
AddressEntryFilter	IMAPITable
AddressList	IAddrBook
Attachment	IAttach
Field	IStream, IMAPIProp
Folder	IMAPIFolder
InfoStore	IMsgStore
Message	IMessage
MessageFilter	IMAPITable
Recipient	IMAPIProp
Session	IMAPISession

For collection objects, the OLE Messaging Library calls the MAPI interface **IMAPITable**.

The OLE Messaging Library also calls the MAPI interface **IMAPIProp**. Many of the properties exposed by the OLE Messaging Library are based on MAPI properties. The following table describes the mapping between these OLE Messaging Library properties and the underlying MAPI properties.

OLE Messaging Library object	Property	MAPI property	MAPI property type
AddressEntry	Address	PR_EMAIL_ADDRESS	PT_TSTRING
AddressEntry	DisplayType	PR_DISPLAY_TYPE	PT_LONG
AddressEntry	ID	PR_ENTRYID	PT_BINARY
AddressEntry	Name	PR_DISPLAY_NAME	PT_TSTRING
AddressEntry	Type	PR_ADDRTYPE	PT_TSTRING
AddressList	ID	PR_ENTRYID	PT_BINARY
AddressList	Name	PR_DISPLAY_NAME	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
InfoStore	ID	PR_ENTRYID	PT_BINARY
InfoStore	Name	PR_DISPLAY_NAME	PT_TSTRING
InfoStore	ProviderNam e	PR_PROVIDER_DISPL AY	PT_TSTRING
Message	Conversation	PR_CONVERSATION_ KEY	PT_BINARY
Message	Conversation Index	PR_CONVERSATION_ INDEX	PT_BINARY
Message	Conversation Topic	PR_CONVERSATION_ TOPIC	PT_STRING
Message	Delivery Receipt	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	Time Received	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	DisplayType	PR_DISPLAY_TYPE	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 MAPI properties, see the *MAPI Programmer's Reference*.

Additional References

The following references provide additional information about OLE and Automation:

- *OLE Programmer's Reference* in the Microsoft Win32 Software Development Kit.
- *Automation* in the Win32 SDK.
- *Inside OLE, Second Edition*, by Kraig Brockschmidt, published by Microsoft Press.

Note that this document contains the latest known information about the Microsoft OLE Messaging Library at the time of publication. Where terms in this document differ from other Visual Basic, OLE, or Component Object Model (COM) terms, this document should be viewed as the definition of the specific implementation represented by the OLE Messaging Library.