

Legal Information

Microsoft Collaboration Data Objects Programmer's Reference

Information in this document is subject to change without notice. This document is provided for informational purposes only and Microsoft Corporation makes no warranties, either express or implied, in this document. The entire risk of the use or the results of the use of this document remains with the user. The names of companies, products, people, characters, and/or data mentioned herein are fictitious and are in no way intended to represent any real individual, company, product, or event, unless otherwise noted. Complying with all applicable copyright laws is the responsibility of the user. No part of this document may be reproduced or transmitted in any form or by any means, electronic or mechanical, without the express written permission of Microsoft Corporation. If, however, your only means of access is electronic, permission to print one copy is hereby granted.

Microsoft may have patents or pending patent applications, trademarks, copyrights, or other intellectual property rights covering subject matter in this document. The furnishing of this document does not give you any license to these patents, trademarks, copyrights, or other intellectual property rights except as expressly provided in any written license agreement from Microsoft.

© 1996-1998 Microsoft Corporation. All rights reserved.

ActiveX, Active Platform, Active Server, Microsoft, JScript, Visual Basic, Visual C++, Windows, Windows NT, and Win32 are trademarks or registered trademarks of Microsoft Corporation in the United States and/or other countries.

Java is a trademark of Sun Microsystems, Inc.

Macintosh is a registered trademark of Apple Computer, Inc. used under license.

Other product and company names mentioned herein may be the trademarks of their respective owners.

Overview of CDO

Microsoft® Collaboration Data Objects (CDO) is a technology for building messaging or collaboration applications. The current version of CDO is 1.2.1. In versions previous to 1.1, CDO was called OLE Messaging; in version 1.1 it was called Active Messaging. It is designed to simplify the creation of applications with messaging functionality, or to add messaging functionality to existing applications. For example, CDO and [Active Server Pages](#) enable you to add script to a Web site to provide support for creating, sending, and receiving e-mail as well as participating in discussions and other [public folder](#) applications.

CDO does not represent a new messaging model, but rather an additional scripting interface to the Messaging Application Programming Interface ([MAPI](#)) model.

CDO is made available through the two [CDO libraries](#). They are described in detail in [Introduction](#) and [Overview of CDO Rendering](#).

These libraries expose programmable messaging objects (including [folders](#), [messages](#), [recipient](#) addresses, [attachments](#), and other messaging components), which are extensions to the programmable objects offered as part of Microsoft® Visual Basic®, such as forms and controls.

Intended Audience

To develop messaging-enabled applications using CDO, your background should include:

- Proficiency with a scripting language, such as Microsoft® Visual Basic® Scripting Edition (also called VBScript), Microsoft® JScript™, or JavaScript.
- Some understanding of messaging technology.

To develop messaging-enabled applications for the World Wide Web, you should also be proficient in the use of Hypertext Markup Language ([HTML](#)). No experience with C/C++ is necessary, although experience with Visual Basic or Visual Basic for Applications is recommended.

Software Requirements

The dynamic-link libraries for CDO 1.2.1 (CDO.DLL) and CDO 1.2.1 Rendering (CDOHTML.DLL) are MAPI client object libraries. As such, they require MAPI (the version of MAPI32.DLL installed with Microsoft® Exchange Server version 5.0, or above) and service providers—for example, the same service providers as with Microsoft Exchange Server. CDOHTML.DLL and CDO.DLL are included with Microsoft Exchange Server and are installed on the server when the installation option **Active Server Components** is selected. CDO 1.2.1 Rendering (CDOHTML.DLL) is intended for server-side use only.

Microsoft® Outlook™ 98 installs the CDO 1.2.1 libraries. To enable client applications to use CDO 1.2.1, install Microsoft Outlook 98. For server applications, install Microsoft Exchange Server version 5.5.

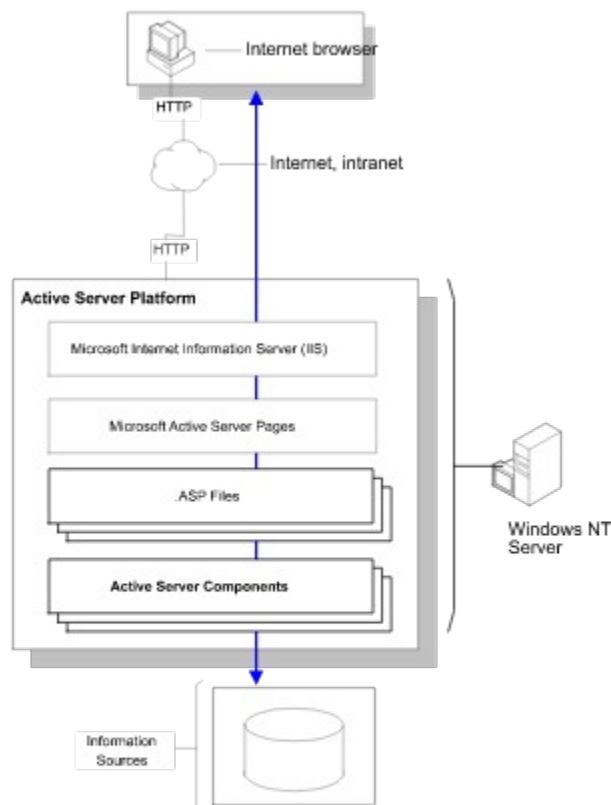
Note CDO works with Microsoft Outlook, but CDO.DLL is not installed with the Microsoft Outlook 97 setup program.

The Active Server Environment

Built into Microsoft® Internet Information Server (IIS) version 3.0, Active Server™ is a server-side and script-based programming model that allows developers to create server and Web server applications.

Active Server also takes full advantage of Microsoft® Windows NT® to create high-performance, scalable Web applications. Shared Internet server applications benefit from a rich set of services for messaging, database access, and transaction support. Because Active Server uses the component model used throughout the Active Platform, third parties can create components and applications that integrate seamlessly with other platform technologies and applications.

CDO in Microsoft® Exchange Server 5.0 is an Active Server component and provides messaging and collaboration functionality to all Active Platform™ applications. The following illustration shows how Active Server components (including, for example, the Active Server Components of Microsoft Exchange) fit within the Active Server platform.



The Active Server Components of Microsoft Exchange

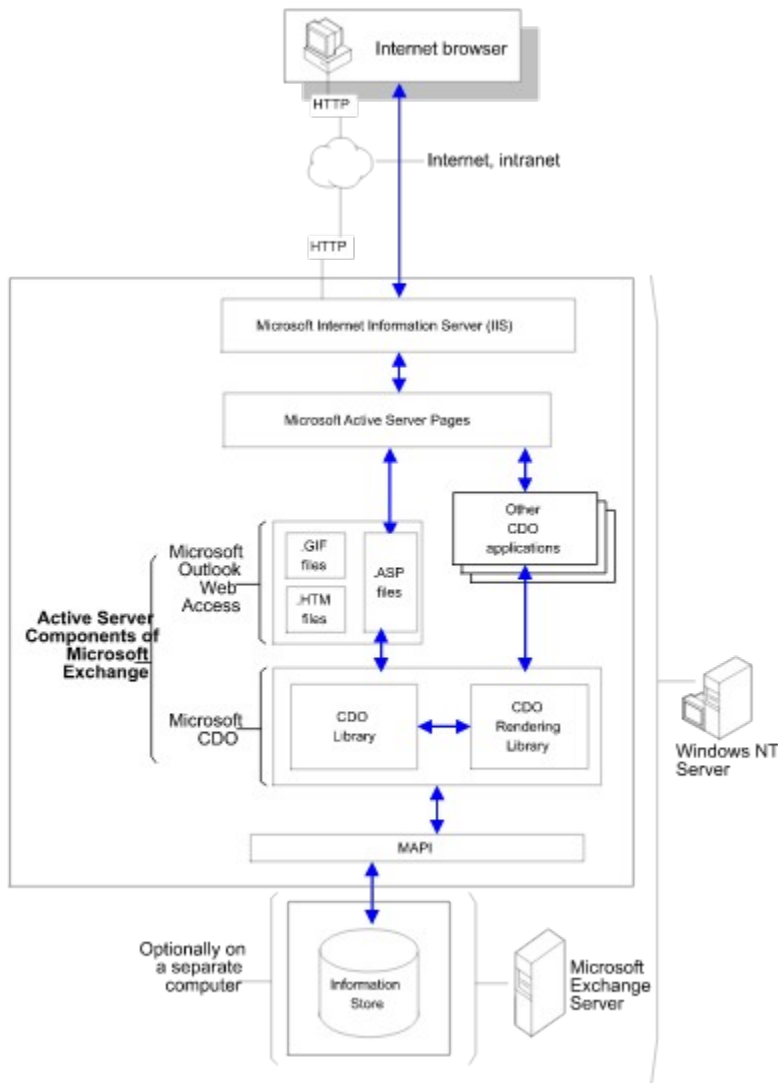
Microsoft® Exchange Active Server components provide the technology for integrating Microsoft Exchange Server with Web applications. This integration provides Web access to Microsoft Exchange Server data using IIS and Active Server Pages (ASP).

With Microsoft Exchange Active Server components, developers create applications by combining scripting, HTML, and core Microsoft Exchange technologies such as messaging, calendaring, and groupware. The Microsoft Exchange Active Server components let developers leverage not only the services of Microsoft Exchange Server but also other Microsoft and third-party services using any Web browser as a front end.

Microsoft Exchange Server installs the Active Server components of Microsoft Exchange if the installation option **Active Server Components** is selected. This installation gives you:

- **Microsoft Collaboration Data Objects**, which includes the CDO Library and the CDO Rendering Library.
- **Microsoft Outlook Web Access**.

As indicated in the following illustration, Microsoft Exchange Server can be installed on the same computer as the Active Server components or on a different computer.



Other elements in this illustration are:

- **Internet browsers** Microsoft® Internet Explorer version 3.0 or Netscape Navigator version 3.0 is required to access Microsoft Exchange ASP-based applications.
- **Microsoft Internet Information Server (IIS)**, a Web server integrated into Windows NT Server. As shown in the illustration, IIS is required to access Microsoft Exchange ASP-based applications.
- **Microsoft Active Server Pages**
- **Microsoft CDO libraries**
- **Microsoft Outlook Web Access**
- **Microsoft Exchange Server**, a client/server messaging and workgroup system that offers a transparent connection to many different communications systems. The two key elements of Microsoft Exchange Server that are used by the CDO libraries are the information store (for accessing mailboxes and public folders) and the directory (for accessing the address book).

Note For information on installing the Microsoft Exchange Active Server components, see the release notes for Microsoft Exchange Server 5.0. For a thorough description of the use of Microsoft® Outlook™ Web Access, see the *What's New* guide for Microsoft Exchange Server 5.0.

About Active Server Pages

Active Server Pages (ASP) is an open, compile-free application environment in which you can combine HTML pages, scripts, and Microsoft® [ActiveX](#)® server components to create powerful Web-based applications.

Active Server Pages offers native support for Microsoft® Visual Basic® Scripting Edition (VBScript) and Microsoft® JScript™, and supports other scripting languages such as REXX, Python, and Perl through Active Scripting plug-ins. Active Server Pages also supports ActiveX Scripting, allowing virtually any scripting engine to be used. It allows Web developers to write scripts that are executed on either the server or the client.

Active Server Pages also supports ActiveX components developed in any language, such as C++, Visual Basic, Java, COBOL, and others. The resulting applications are compatible with any Web browser running on any operating system. (The server-side scripting processor also allows for [multilingual support](#), defining in which language HTML is returned, based on user preference.)

Script Processing

Microsoft Active Server Pages processes ASP scripts. When Active Server Pages encounters regular HTML text, it does not process it, but passes it through the Active Server response object (and an **IStream** object) directly to the browser. When Active Server Pages encounters text within server-side script tags (<% and %>), it processes this script code and generates HTML, which it sends to the browser.

Scripts are not stored in a compiled form. Rather, they are interpreted when the ASP file is requested from the server.

Active Server Pages Sessions

Active Server Pages is not stateless. This means that, to use it, you need to start an Active Server Pages [session](#). Such a session exists – and a Session object is created – after a user connects. When the session expires, the Session object is destroyed. In contrast, the Hypertext Transfer Protocol ([HTTP](#)) is stateless, which means that no "state" information concerning the requester is maintained between successive requests to the HTTP server.

Because the CDO libraries are an interface to MAPI, they must use a MAPI session. A MAPI session object is stored as a state variable in an Active Server Pages session, which means that a user need not log on to MAPI for each request.

Active Server Pages uses a default twenty-minute time-out. Scripts used with [Microsoft Outlook Web Access](#) set this time-out to sixty minutes for authenticated users. When this time-out expires, the Active Server Pages session and the MAPI session objects are destroyed.

About ASP Files

Active Server Pages (.asp) files are standard HTML documents interlaced with ActiveX script code that calls specific Active Server components, such as CDO.

The Microsoft Exchange Active Server components include a number of server-side scripts in .asp files. The scripting in these .asp files functions through calls to the interfaces in the CDO Library and the CDO Rendering Library. Active Server Pages processes these .asp files, and uses an **IStream** interface to send to the browser either generated HTML content or a new (changed) .asp file, which the browser interprets.

Some .asp files are specific to [Microsoft Outlook Web Access](#), and others are useful for other ASP-based collaboration applications.

Because .asp files are not compiled and the ASP source code for Web access to Microsoft Exchange Server is included, the source files for Microsoft Outlook Web Access can be used as sample code for building other applications or can be modified to customize its behavior, visual appearance, or functionality. Because Microsoft Exchange Active Server components are built on the Active Server platform using the [CDO libraries](#), they illustrate how other collaborative Web applications can be created using CDO.

For more information about customizing .asp files for use with Microsoft Exchange Server, see [Overview of CDO Rendering](#).

About the CDO Libraries

The CDO and CDO Rendering libraries are used for building collaborative Web server applications on Microsoft® Exchange Server. The CDO Library can be used to build both client and server applications, but the CDO Rendering Library can be used only for server applications.

The CDO libraries can be called from Microsoft® Visual Basic® or Visual Basic Scripting Edition (VBScript), Microsoft® JScript™, Javascript, Java, and any application that supports Visual Basic for Applications such as Microsoft® Office.

These libraries are used by the following classes of applications:

- Server applications integrated with Active Server Pages and a browser to provide Web access to client features, namely mailboxes, and public folders.
- Client applications for client-side scripting. In this case, the CDO libraries are not used with Active Server Pages.

Note Scripts can be run on the server or on the client. The .asp files for Microsoft® Outlook™ Web Access include scripts written in VBScript and executed by Microsoft® Internet Information Server (IIS); client-side scripts are written in VBScript, JScript, or JavaScript and executed by the browser.

These are the two CDO libraries:

- **CDO Library** This library lets you add to your 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.
- **CDO Rendering Library** This library is used to render CDO objects and collections in HTML for use on the World Wide Web.

Note If your purpose is to run on a Web server to expose content to the Internet, you need a computer running IIS to work with the CDO libraries. However, you can use the CDO Library alone on the client (or on the server) as a scripting library. In this case, it is used as a general-purpose library, and IIS is not required.

Introduction

The Microsoft® Collaboration Data Objects (CDO) Library version 1.2.1 exposes messaging objects for use by Microsoft® Visual Basic®, C/C++, and Microsoft® Visual C++® applications. (In versions previous to 1.1, the CDO Library was called the OLE Messaging Library; in version 1.1 it was called the Active Messaging Library.)

The CDO 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 CDO 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 CDO Library, this guide provides a short overview of the MAPI architecture. For complete reference information, see the *MAPI Programmer's Reference*.

The CDO Library requires installation of MAPI and of an automation controller. An automation controller is an application that supports Automation, such as 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 Microsoft® 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.

' This sample uses Visual Basic 3.0 error handling.

```
Function QuickStart()  
Dim objSession As MAPI.Session ' use early binding for more efficient  
Dim objMessage As Message      ' code and type checking  
Dim objOneRecip As Recipient  
  
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:="Sender Name"  
  
' create a message and fill in its properties  
Set objMessage = objSession.Outbox.Messages.Add  
objMessage.Subject = "Sample Message"  
objMessage.Text = "This is sample message text."  
  
' create the recipient  
Set objOneRecip = objMessage.Recipients.Add  
objOneRecip.Name = "Recipient Name"  
objOneRecip.Type = CdoTo  
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 CDO Library invalidates the Message object after you call its **Send** method. This code fragment logs off to end the session after sending the message, but if you continued the MAPI session, you could avoid potential errors by setting the Message object to **Nothing**.

About Installation

The Collaboration Data Objects (CDO) Library version 1.2.1 is installed with the Microsoft® Exchange Client and Server, and with Microsoft® Outlook™ 98. The setup programs register the CDO Library for subsequent use by automation controllers, that is, applications that support Automation.

Note Microsoft® Outlook™ version 8.03 includes the predecessor of CDO version 1.2, namely Active Messaging 1.1.

When you use the CDO Library with an automation controller, verify that the tool has referenced the CDO 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 **Microsoft CDO 1.2.1 Library**.

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

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

The **OLEMsgPersistenceTimeout** registry setting controls how quickly the CDO 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 Document

Overview discusses:

- MAPI terms used in this guide;
- comparison between the CDO Library and the other MAPI programming interfaces;
- Automation and the operation of CDO within this technology;
- the design of the CDO Library, including the objects and collection objects that are available to you.

Programming Tasks offers sample Microsoft® 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.

Designing and Creating Forms discusses forms and their relationship to other e-mail messages. It then explains how to create forms, and concludes with an extended example of form creation.

Objects, Properties, and Methods provides a summary of every object and collection object in the CDO library, listing all the properties and methods each one exposes. This is followed by a model of all the objects presented in their containment hierarchy.

Properties Common to All CDO Library Objects describes the properties that are exposed by every object in the CDO library and which have the same meaning for every object.

This is followed by an alphabetical series of topics covering every object and collection in the CDO Library and providing comprehensive reference information for their properties and methods.

The appendixes offer additional background information about various aspects of CDO programming:

- Error Codes describes and lists the warning and error codes that can be returned from various properties and methods.
- MAPI Property Tags lists the property tags and type library constants for predefined MAPI properties.
- Microsoft Exchange Property Tags lists the property tags for predefined Microsoft® Exchange properties.
- Web Page Support describes how the CDO Library provides support for HTML (Hypertext Markup Language) script on a Web page.
- How Programmable Objects Work provides a detailed explanation of COM objects, Automation, and the relationship between CDO and MAPI.
- Java Programming Considerations discusses procedures and provides examples for accessing CDO Library objects from Java.
- Common Mistakes discusses design and coding errors that can be made when programming with the CDO Library.

The document concludes with a Glossary, which defines terms that are important to understanding CDO, MAPI, Automation, COM, and other related technologies.

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

Overview

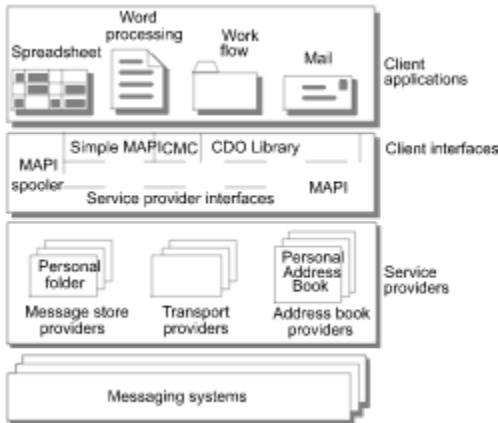
This section presents a brief introduction to MAPI and describes how the Collaboration Data Objects (CDO) Library fits into the mix of MAPI programming interfaces. It provides a short description of Automation, which is the basis of the design of the CDO Library. The section also offers a conceptual overview of the CDO Library, and a discussion of differences between 16-bit and 32-bit platforms.

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. 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, an e-mail type, and an 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.

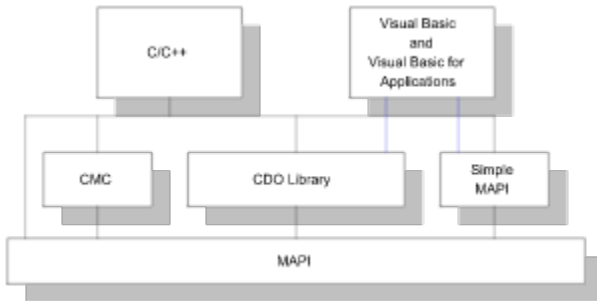
Several *properties* can be associated with a message: its subject, its importance, its delivery properties (such as the time it is sent and received), and whether to notify the sender when the message is delivered and read. Some message properties identify the message as part of a *conversation*. The 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 CDO 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 CDO Library is available to all the concerned languages, namely Microsoft® Visual Basic®, Visual Basic for Applications, and C/C++.



It is important to recognize that the CDO 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 later, Visual Basic for Applications, 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.
CDO Library	Programmable messaging objects for Visual Basic/Visual Basic for Applications 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 CDO Library

Although both the MAPI custom controls and the CDO 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 CDO Library lets you add messaging to your applications.

MAPI Functions and the CDO 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 CDO 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 CDO Library interface.

Task or code	Function call interface	CDO 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 <u>Message</u> 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 unwieldy. In contrast, the CDO 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 CDO Library is based on the capabilities provided by Automation. The CDO Library allows you to create instances of programmable messaging *objects* that you can reference with automation controllers. An *automation controller* is a tool that supports Automation, such as Microsoft® 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 CDO Library.

Throughout this document, Visual Basic is used as a concrete example of an automation controller, but the statements about Visual Basic apply to all such tools.

Visual Basic scripts drive the CDO 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 CDO Library objects to create a message containing the results of those calculations and send the message to several recipients.

CDO Library Object Design

The CDO Library is designed for ease of use and convenience. It implements the MAPI functions most used by client applications. The CDO Library is not designed for development of service providers. (For more information about service providers, see [Introduction to MAPI](#).)

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

The CDO Library defines the following objects:

[AddressEntries](#) collection

[AddressEntry](#)

[AddressEntryFilter](#)

[AddressList](#)

[AddressLists](#) collection

[Attachment](#)

[Attachments](#) collection

[Field](#)

[Fields](#) collection

[Folder](#)

[Folders](#) collection

[GroupHeader](#)

[InfoStore](#)

[InfoStores](#) collection

[Message](#)

[MessageFilter](#)

[Messages](#) collection

[Recipient](#)

[Recipients](#) collection

[Session](#)

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

[Top-level objects](#), which can be created directly in a Microsoft® Visual Basic® program.

[Child objects](#), which can be instantiated under the top-level objects.

[Collections](#), or groups of objects of the same type.

Top-Level Objects

A top-level object is one that can be created directly by your code, without having to derive it from any other object. Currently, the only top-level object in CDO is the Session object. Other objects are accessible only through the Session object.

You can create a Session object either through early binding:

```
Dim objSession As MAPI.Session
Set objSession = CreateObject ("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. You cannot access any other object, or even the Session object's properties or methods, until you log on. The only exception to this rule is the Session object's SetLocaleIDs method.

Generally, early binding is preferable, because it enforces type checking and generates more efficient code. Note that you specify "MAPI.Session" instead of just "Session" in order to distinguish a MAPI session from other types of sessions available to a Visual Basic program through other object libraries.

Early binding is not supported in CDO Library versions previous to 1.1.

C/C++ programmers use globally unique identifiers (GUIDs) for these objects, defined in the type library for the CDO 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 CDO 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 CDO 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.IVal);
// 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 top-level objects accessible to C/C++ programmers.

CDO Library object	GUID
<u>Session</u>	{3FA7DEB3-6438-101B-ACC1-00AA00423326}

Child Objects

All CDO 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 CDO 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 CDO Library, the name of the collection takes the plural form of the individual CDO Library object. For example, the Messages collection is the name of the collection that contains Message objects. The CDO 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 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 CDO Library).

Small collections, with a known number of member objects, have a reliable **Count** property, which always contains the current number of member objects. The **Item** property can be used to select any arbitrary member of the collection. A small collection also has an implicit temporary **Index** property, assigned by the CDO 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. An 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 the objects in the collection. 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** property in a limited way. If the value of **Count** is set to **CdoMaxCount**, the provider is unable to furnish an accurate number of members, or even to indicate whether the collection is empty or not. If **Count** has a value other than **CdoMaxCount**, its value is reliable. The **Item** property has the same functionality as it does in small collections. For more information on using the **Count** and **Item** properties in a large collection, see the example in the [Count](#) property.

The **Count** property is updated whenever you refresh an [AddressEntries](#) or [Messages](#) collection, in particular by altering its child [AddressEntryFilter](#) or [MessageFilter](#) object.

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** method, specifying the unique identifier, to obtain the individual AddressEntry, Folder, or Message object. You can also use the **GetFirst** and **GetNext** methods to move from one object to the next in these collections.

Note To ensure correct operation of the **GetFirst**, **GetLast**, **GetNext**, and **GetPrevious** methods in a large collection, call **GetFirst** before making any calls to **GetNext** on that collection, and call **GetLast** before any calls to **GetPrevious**. To ensure that you are always making the calls on the same collection, create an explicit variable that refers to that collection.

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

```
' fragment 1: each Set statement creates a new Messages collection;  
'           it's undefined which message is returned by GetNext  
Set objMessage = objInBox.Messages.GetFirst  
...  
Set objMessage = objInBox.Messages.GetNext
```

```
' fragment 2: use an explicit variable to refer to the collection;  
'           now the Get methods return the intended messages  
Set objMsgColl = objSession.objInBox.Messages  
Set objMessage = objMsgColl.GetFirst  
...  
Set objMessage = objMsgColl.GetNext
```

Code fragment 1 causes the CDO Library to create a new Messages collection in each **Set** statement. The **GetFirst** call returns the first message in the collection, but the result of the **GetNext** call is undefined since **GetFirst** has not yet been called on this new collection.

Code fragment 2 creates and uses the explicit variable *objMsgColl*, so the **GetFirst** and **GetNext** calls act as expected for collections with more than one item.

The collections in the CDO 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 CDO Library supersedes the other documentation.

Programming Tasks

This section describes some of the common programming tasks you can perform with the Collaboration Data Objects (CDO) Library. The first task your application must complete is to obtain and **Logon** to a valid Session object as described in Starting a CDO Session. The following table categorizes the described tasks:

Category	Programming tasks
General programming tasks	<u>Handling Errors</u> <u>Improving Application Performance</u> <u>Starting a CDO Session</u> <u>Viewing MAPI Properties</u>
Working with messages	<u>Adding Attachments to a Message</u> <u>Checking for New Mail</u> <u>Creating and Sending a Message</u> <u>Customizing a Folder or Message</u> <u>Deleting a Message</u> <u>Filtering Messages in a Folder</u> <u>Making Sure the Message Gets There</u> <u>Reading a Message from the Inbox</u> <u>Searching for a Message</u>
Working with addresses	<u>Changing an Existing Address Entry</u> <u>Creating a New Address Book Entry</u> <u>Selecting Recipients from the Address Book</u> <u>Using Addresses</u> <u>Working with Distribution Lists</u>
Working with folders	<u>Accessing Folders</u> <u>Copying a Message to Another Folder</u> <u>Customizing a Folder or Message</u> <u>Moving a Message to Another Folder</u> <u>Searching for a Folder</u>
Working with public folders	<u>Posting Messages to a Public Folder</u> <u>Working with Conversations</u>

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

Programming task	Procedure
<u>Accessing Folders</u>	<ol style="list-style-type: none">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.
<u>Adding Attachments to a</u>	<ol style="list-style-type: none">1. Create or obtain the Message object that

Message

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 Address Entry

1. Obtain a valid AddressEntry object.
2. Update the AddressEntry object's **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 AddressEntries 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 is to 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 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 CDO 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** properties 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.

Organizing a Meeting

1. Obtain a calendar folder from the session.
2. Add an appointment to the calendar folder.
3. Create a Recipients collection for the appointment.
4. Populate the appointment's Recipients

collection.

5. **Send** the appointment to the recipients.

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.

Selecting Recipients from

1. Call the session's **AddressBook** method to use the MAPI address book dialog box.

the Address Book

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 a CDO 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**.

Working with Distribution Lists

1. Add a distribution list (DL) to a personal address book (PAB).
2. Add an address entry to a distribution list.
3. Delete an address entry from a distribution list.

It is important to understand the hierarchy of the CDO Library objects, because the hierarchical relationships between objects determines the correct syntax of Microsoft® 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. For more information on the hierarchy, see [Object Model](#).

In the sample code that appears in this guide, individual statements are often broken across several lines. The convention used for this is the statement continuation introduced in Visual Basic version 4.0, which consists of a space followed by the underscore character (_). This sequence is placed at the end of a code line to indicate that the current statement is continued on the next line.

Accessing Folders Group

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 CDO 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"  
    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 CDO Library 1.1, can use Class value:
    ' If CdoFolder = 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

Group

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 CDO 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
CdoFileData	<u>ReadFromFile</u> method
CdoFileLink	Source property
CdoOLE	ReadFromFile method
CdoEmbeddedMessage	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 a CDO Session.

```
' Function: Attachments_Add_Data
' Purpose: Demonstrate the Add method for type = CdoFileData
' 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 = CdoFileData
        .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 CdoFileData 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 CDO 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 by the **Position** property.

To insert an attachment of type **CdoOLE**, use code similar to the **CdoFileData** type example. Set the attachment type to **CdoOLE** 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 **CdoFileLink**, set the **Type** property to **CdoFileLink** 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 = CdoFileLink
' 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 = CdoFileLink
        .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 CdoFileLink"

```

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

Group

The CDO 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 Group

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 CDO 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

Group

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

Note With versions of CDO Library previous to 1.1, 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 capability of every property is also preserved across the copy.

Group

To copy a message from one folder to another folder using the CDO 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 a CDO 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 ' to 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

Group

You can create new address entries in a collection with the CDO 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. After 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 PAB. Note the use of early binding and of default properties. The objects are declared using early binding to force matching of object types, and to distinguish a MAPI session from other types of sessions available to a Microsoft® Visual Basic® program through other object libraries. 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
Dim propTag As Long           ' MAPI property tag for new field
```

```
On Error GoTo error_olemsg
Set objSession = CreateObject("MAPI.Session")
```

```
' 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 = objSession.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 (don't need to Add it)  
propTag = &H3A08001E ' VB4.0: CdoPR_BUSINESS_TELEPHONE_NUMBER  
objNewEntry.Fields(propTag) = "+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

Group

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

Group

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 ' or Dim objSession As MAPI.Session
Dim objMessage As Object ' or Dim objMessage As Message
Dim objOneRecip As Object ' or Dim objOneRecip As Recipient

    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:="Jane Doe", _
                'profilePassword:="my_pword"

' create a message and fill in its properties
Set objMessage = objSession.Outbox.Messages.Add
objMessage.Subject = "Sample Message"
objMessage.Text = "This is some sample message text."

' create the recipient
Set objOneRecip = objMessage.Recipients.Add
objOneRecip.Name = "John Doe"
objOneRecip.Type = CdoTo
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.

You can send a message on behalf of another messaging user by setting the **Sender** property twice. For more information, see the Sender property of the Message object.

See Also

[Adding Attachments to a Message](#), [Customizing a Folder or Message](#)

Customizing a Folder or Message

Group

The CDO 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 CDO 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, suppose 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
```

Note 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 CDO 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

Group

The [Message](#) object's [Delete](#) method deletes the message.

Group

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

Group

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 object. 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 February 6, 1998 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 ( "02/06/98" )
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 Group

The CDO Library raises exceptions for all errors. When you write Microsoft® Visual Basic® applications that use the CDO 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 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 earlier versions of Visual Basic, the error value is returned as the sum of decimal 1000 and the low-order word of **HRESULT**. This is because versions of Visual Basic previous to 4.0 reserve all run-time error values below 1000 for their own errors.

This code fragment checks for an error corresponding to the MAPI error code **CdoE_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 &H0113, 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
Dim objSession As MAPI.Session
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 = CdoE_USER_CANCEL Then
    MsgBox "User pressed Cancel"
Else
    MsgBox "Unrecoverable Error:" & Err
End If
Util_CreateSessionAndLogon = False
Exit Function

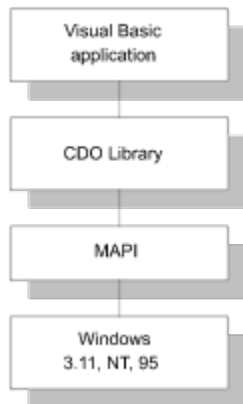
error_olemsg:
MsgBox "Error " & Str(Err) & ": " & Error$(Err)
Resume Next

End Function

```

When an error occurs in the MAPI subsystem, the CDO 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 CDO Library. Visual Basic applications reside at the highest level and interact with the CDO Library at the next lower level. The CDO 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 can display the complete **HRESULT** or error code value returned by the CDO Library.

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

See Also

[Starting a CDO Session](#)

Improving Application Performance

Group

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

To access CDO 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 CDO 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 CDO Library to create an internal Folder object that represents the user's Inbox. The next portion, `.Messages`, directs the CDO Library to create an internal Messages collection object. The final part, `.GetFirst`, directs the CDO Library to create an internal Message object that represents the first message in the user's Inbox. The statement contains three periods; the CDO 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 CDO 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 are 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 CDO 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 CDO Library creates a temporary internal object that represents the Messages collection, then discards it after displaying the first message. The second statement directs the CDO 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 Visual Basic programming documentation.

Making Sure the Message Gets There

Group

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.

Moving a Message to Another Folder

Group

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 CDO 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.

Group

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.

Organizing a Meeting Group

A meeting is an appointment applicable to one or more messaging users. If you wish to organize a meeting, you start with an appointment that specifies the time and place, and you send it to the other users that are to be involved.

Group To organize a meeting using the CDO Library

1. Create a valid session and log on.
2. Obtain a calendar folder from the active session.
3. Create a new AppointmentItem object in the calendar folder using its Messages collection's **Add** method.
4. Set the appointment's MeetingStatus property to **CdoMeeting**.
5. Set the StartTime and EndTime properties on the appointment, and other properties as applicable, such as Location, Subject, and Text.
6. Instantiate a Recipients collection for the appointment using its Recipients property.
7. **Add** the appropriate messaging users to the Recipients collection.
8. Set other properties on the AppointmentItem and Recipient objects as appropriate.
9. **Send** the appointment to the selected recipients.

The hierarchy of relevant objects is as follows:

```
Session object
  Folder object
    Messages collection
      AppointmentItem
        Recipients collection
          Recipient
          RecurrencePattern
        MeetingItem
```

You create a meeting by setting the AppointmentItem object's MeetingStatus property to **CdoMeeting** and sending it to one or more recipients. The appointment becomes a meeting as of the moment you call its **Send** method. In response to this, CDO automatically instantiates a MeetingItem object in your Inbox and every Recipient object's Inbox. Each MeetingItem object becomes a member of the Inbox Messages collection, and it can be treated programmatically just like the regular Message objects in the collection.

To accept a meeting request, a recipient calls the MeetingItem object's **Respond** method with **CdoResponseAccepted** in the *RespondType* parameter. **Respond** returns another MeetingItem object, which the recipient can **Send** back to the organizer. Calling **Respond** with either **CdoResponseAccepted** or **CdoResponseTentative** causes an associated AppointmentItem object to be created in the recipient's active calendar folder.

A recipient can call the MeetingItem object's GetAssociatedAppointment method to obtain the associated AppointmentItem object. **GetAssociatedAppointment** returns the appointment created by CDO in that recipient's calendar folder, not the original appointment from which you created the meeting. You are identified in the Organizer property of every recipient's associated appointment.

To decline a meeting request, a recipient calls **Respond** with *RespondType* set to **CdoResponseDeclined**, and then **Send**. No AppointmentItem object is created in the recipient's calendar folder in this case.

If you decide to cancel a meeting, set the original appointment's MeetingStatus property to **CdoMeetingCanceled** and **Send** it again to all the recipients. Only you can cancel a meeting that you

organized.

You can make an appointment recurring by calling its **GetRecurrencePattern** method, which returns a child **RecurrencePattern** object describing the recurrence characteristics. You can then change the **RecurrencePattern** object's properties from their default values to specify the recurrence settings you want. The appointment can later be restored to nonrecurring status with the **ClearRecurrencePattern** method. It is usually best to make an appointment recurring before generating a meeting from it.

If you wish to edit an individual recurrence of a recurring appointment to make it different from other recurrences in the series, you must first instantiate it. To do this, you perform the following steps:

1. instantiate a **MessageFilter** object on the **Messages** collection using its **Filter** property;
2. access the filter's **Fields** collection through its **Fields** property;
3. add **Field** objects with property tags **CdoPR_START_DATE** and **CdoPR_END_DATE**, using the **Fields** collection's **Add** method;
4. set these fields to the start and end date/time of the desired recurrence;
5. call the **Messages** collection's **GetFirst** method to obtain an **AppointmentItem** object representing the desired recurrence.

A meeting can be obtained from its parent **Messages** collection using the collection's **Item** property. To get to the **Messages** collection in a folder, use the **Folder** object's **Messages** property. If you know a meeting's unique identifier, you can obtain it directly from the **Session** object's **GetMessage** method.

This code fragment creates and sends an appointment for a meeting at 9:00 the next morning:

```
Dim objSess As Session
Dim objCalFold As Folder ' calendar folder to create appointments in
Dim colAppts As Messages ' appointments collection in calendar folder
Dim objNewAppt As AppointmentItem
Dim colRecips As Recipients ' users to be invited to the meeting

Set objCalFold = objSess.GetDefaultFolder(CdoDefaultFolderCalendar)
Set colAppts = objCalFold.Messages
Set objNewAppt = colAppts.Add ' no parameters when adding appointments
With objNewAppt
    .Importance = CdoHigh
    .Subject = "Extremely important meeting tomorrow!"
    .Text = "We will discuss stock distribution."
    .StartTime = DateAdd("d", 1, DateAdd("h", 9, Date))
    .EndTime = DateAdd("h", 1, .StartTime) ' meeting to last one hour
    Set colRecips = .Recipients ' empty collection of users to invite
End With
With colRecips ' populate collection with invitees
    .Add "John Doe", "jdoe@mycompany.com" ' can't use parentheses!
    .Add "Mary Coe", "mcoe@mycompany.com"
    .Resolve ' default to dialog in case of name ambiguities
End With
objNewAppt.Send ' appointment becomes a meeting at this moment
```

Posting Messages to a Public Folder

Group

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 and Unread properties to **True**, the Submitted property to **False**, and the TimeReceived and TimeSent properties to the current time.

When you are ready to make the message available, call the Update method. The message is not accessible by any other messaging user until you call **Update**.

You can identify yourself as the poster of the message by setting its **Sender** property. You can also post the message on behalf of another messaging user by setting the **Sender** property twice. For more information, see the Sender property of the Message object.

For more information on sending messages, see Creating and Sending a Message.

Group

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 and Unread properties to **True**, and the Submitted property to **False**.
4. Call the Message object's 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 with the **Update** method, your application must set the time properties. Set both time properties to the same value, just before you set the **Sent** 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 car wanted"  
.Text = "Wanted: Used car in good condition with low mileage."  
.ConversationTopic = .Subject  
.ConversationIndex = Util_GetEightByteTimeStamp() ' utility  
.TimeReceived = Time  
.TimeSent = .TimeReceived  
.Sent = True  
.Submitted = False  
.Unread = True  
.Update ' .Send is not used for posting to a folder  
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

Group

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 [CDO Library Object Design](#), the CDO 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 a CDO Session](#):

```
Dim objSession As MAPI.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 Microsoft® 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 Group

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"
    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 CDO 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 Group

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"
    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 CDO 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 CDO 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](#)

Selecting Recipients from the Address Book Group

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 [CDO Library Object Design](#), the CDO 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 a CDO 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:="&Cdo") ' 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 a CDO Session Group

As described in [CDO Library Object Design](#), all messaging objects are relative to the [Session](#) object. The first task of every application is to create a valid Session object and call its [Logon](#) method. No other method or property of the Session object can be accessed, and no other CDO Library object can be created, until the application has successfully logged on. The only exception to this rule is the Session object's [SetLocaleIDs](#) method.

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

```
Function Util_CreateSessionAndLogon() As Boolean
Dim objSession As MAPI.Session ' use early binding for type checking
On Error GoTo err_CreateSessionAndLogon

Set objSession = CreateObject("MAPI.Session")
' call objSession.SetLocaleIDs here if you need to change your locale
objSession.Logon
Util_CreateSessionAndLogon = True
Exit Function

err_CreateSessionAndLogon:
If (Err = 1275) Then ' VB4.0: If Err.Number = CdoE_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 CDO 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()
Dim objSession As MAPI.Session
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 = CdoE_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 CDO 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 Group

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 CDO 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@company.com".

In contrast to the address book, the objects that you create with the CDO 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:

CDO 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	AddressEntry object's unique identifier
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 object's **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. You can optionally request that **Resolve** display a dialog box.
- 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. You need **Resolve** even though you have supplied the address, because it must be made into an object and given an entry identifier.
- 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.

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:=CdoTo)
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:someone@microsoft.com", _
    Type:=CdoTo)
If objOneRecip Is Nothing Then
    MsgBox "Unable to add recipient using custom addressing"
    Exit Function
End If
objOneRecip.Resolve ' assign entry identifier

' 3. add an existing valid address entry object
Set objOneRecip = objNewRecips.Add( _
    entryID:=strAddrEntryID, _
    Type:=CdoTo)
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 Group

You can use a feature of the CDO Library's [Fields](#) 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 CDO Library also supports *multivalued* properties, or properties that represent arrays of values. A multivalued property appears to the Microsoft® 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 CDO 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 CDO Library works with three types of message properties:

- Standard MAPI properties with property tags defined as constants by the CDO Library, such as **CdoPR_MESSAGE_CLASS**.
- Standard MAPI properties not defined by the CDO 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 exposes standard MAPI properties not defined by the CDO 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 CDO 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 IValue 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
```

```
IValue = &H001A001E ' VB4.0: IValue = CdoPR_MESSAGE_CLASS  
' &H001A = PR_MESSAGE_CLASS; &H001E = PT_TSTRING  
' high-order 16 bits = property ID, low-order = property type  
Set objOneField = objFieldsColl.Item(IValue)  
If objOneField Is Nothing Then  
    MsgBox "Could not get the Field using the value " & IValue  
    Exit Function  
Else  
    strMsg = "Used the value " & IValue & " 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

Group

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 a value appropriate to all messages within the conversation, not only to the first message. 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 IResult As Long
Dim IGuid As GUID
' Exchange conversation is a unique 8-byte value
' Exchange client viewer sorts by concatenated properties
On Error GoTo error_olemsg
```

```

IResult = CoCreateGuid(IGuid)
If IResult = S_OK Then
    Util_GetEightByteTimeStamp = Hex$(IGuid.Guid1) & Hex$(IGuid.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](#).

Working with Distribution Lists

Group

You can work with a distribution list (DL) if you have the appropriate permissions in the address list that contains it. Normally you have the required permissions in your personal address book (PAB), in which you can create and modify a private distribution list (PDL). Permissions for the global address list (GAL) are typically limited to a very few profiles, but users with appropriate permissions can modify it as well.

The following code fragment provides two utility functions: **AddDLToAL**, which creates a new private distribution list in the address list the caller specifies by name; and **AddUserToDL**, which adds an address entry to a distribution list specified by the caller.

```
Public Function AddDLToAL(objAL As MAPI.AddressList, _
                        strDLName As String) As MAPI.AddressEntry
' Create a new (private) DL in the specified address list
  Dim colAEs As MAPI.AddressEntries
  Dim objNewAE As MAPI.AddressEntry
  On Error GoTo Err_AddDLToAL

  Set colAEs = objAL.AddressEntries
  Set objNewAE = colAEs.Add("MAPIPDL", strDLName)
  objNewAE.Update ' commit changes to persistent storage
  Set AddDLToAL = objNewAE ' return added PDL (AddressEntry object)
  Exit Function
Err_AddDLToAL:
  Set AddDLToAL = Nothing ' no object was added to address list
  Exit Function
End Function
```

```
Public Function AddUserToDL(objDL As MAPI.AddressEntry, _
                          strAENAME As String, _
                          strAEAddr As String) As Boolean
' Add an address entry to the specified DL
  Dim objNewAE As MAPI.AddressEntry
  On Error GoTo Err_AddUserToDL

  Set objNewAE = objDL.Members.Add("SMTP", strAENAME)
  With objNewAE
    .Address = strAEAddr
    .Fields(CdoPR_COMMENT) = "Added by AddUserToDL function"
    .Fields(CdoPR_COMPANY_NAME) = "Our company"
    .Fields(CdoPR_DEPARTMENT_NAME) = "Your department"
    .Fields(CdoPR_MANAGER_NAME) = "Your manager"
    .Fields(CdoPR_TITLE) = "Dr/Mr/Mrs/Ms/M/Mme/Mlle/Sr/Sra/Herr/Fr"
    .Fields(CdoPR_GIVEN_NAME) = "Your first name"
    .Fields(CdoPR_SURNAME) = "Your last name"
  .Update
  End With
  AddUserToDL = True ' successful
  Exit Function
Err_AddUserToDL
  AddUserToDL = False ' unsuccessful
  Exit Function
```

End Function

Designing and Creating Forms

A form is a custom e-mail message that, because of its distinct appearance and functionality, is rendered differently than a standard e-mail message. This difference reflects the addition of nonstandard fields used to provide the added functionality.

The way a message is rendered is controlled by its message class. In Microsoft® Exchange, standard e-mail messages have the class IPM.Note, and every new message is assigned this class by default. This means that when a person uses a Web browser to create a message, Collaboration Data Objects (CDO) renders a new message of the class IPM.Note. As with non-Web e-mail clients, the sender can then fill in the message's fields and mail it. It is sent as an IPM.Note and displayed as such to the recipient.

A new message class is typically distinguished by the presence of a different set of message fields, which provide access to MAPI properties. The new class usually contains all the fields of the standard message plus others that make it useful for its specific purpose. Regardless of a form's appearance, function, or message class, it remains a message, and Microsoft Exchange stores and transfers it like any other message.

Creating Web Forms with CDO

Traditionally, the Microsoft Exchange Electronic Forms Designer or Microsoft® Visual C++® is used to create custom forms for use with Microsoft Exchange Server. Forms can also be created with CDO for use with a Web browser, a usage that provides the advantage of platform independence when viewing messages and using forms. Regardless of hardware and operating system, if a Web browser can be used on a given computer, it can render any message or form written using CDO.

Technically, the World Wide Web has no concept of e-mail forms, or even e-mail messages. On the Web, only pages are rendered. Still, each page can contain a set of controls that provide nearly all the functionality offered by a form created with the Electronic Forms Designer or Visual C++.

Active Server Pages renders messages and forms using a collection of ASP scripts. These scripts contain not only the CDO code that handles information received from users, but also the HTML code that interacts with users. This code includes input boxes, option buttons, and list controls that a standard message does not include.

For details about form creation steps, see [Creating the New Classified Ad Form](#).

Message Classes and Microsoft Outlook Web Access

Microsoft® Outlook™ Web Access recognizes a form's message class by reading its **Type** property. It parses this property to determine the path to the folder where the form's .asp files are installed. For example, the message class for the Classified Ad sample is:

```
IPM.Microsoft.ITG.NewClassifiedAd
```

Web Access starts with the string `\exchsrvr\webdata\usa\forms` and appends the **Type** property string, replacing each period with a backslash character. This gives the following folder path:

```
\exchsrvr\webdata\usa\forms\IPM\Microsoft\ITG\NewClassifiedAd
```

When you create a form meant for use with Web Access, follow this convention when choosing where to save its .asp files and where to install them after distribution. Similarly, forms you download should be installed into a folder path that follows this convention.

A separate form exists for creating a purchase offer. This form has the message class `IPM.MS.ITG.MicroNews.PrchOfr`, and the files that render it can be found in the folder `\exchsrvr\webdata\usa\forms\IPM\MS\ITG\MicroNews\PrchOfr`. The existence of this message class illustrates how the use of different message classes lets one set of information – about an item for sale, for example – be presented in different ways. Different forms are rendered for different purposes, though the same data can be used in each form.

Microsoft Outlook Web Access uses this convention in the following way:

Group

Using Web Access to open an existing form

1. At the Web Access interface, a user opens a form by clicking it.
2. Web Access reads the form's **Type** property and uses it to determine the directory path to the form's installed .asp files.
3. Web Access uses that path to locate and run the file called `Read.asp`. The `Read.asp` file can now redirect control to any other .asp or .htm file.
4. Typically, CDO functions are now called that use [MAPI](#) to retrieve properties from the form's message object stored in a Microsoft Exchange information store. This data populates the controls of the form, which are rendered to the user.

Form Example: A Classified Ad

A form for classified advertisements was originally created for use with Microsoft® Exchange Server and has been modified for use on the Web.

The following sections describe design considerations for this sample application and step you through some of its code, which is found in various Active Server Pages (ASP) scripts.

- [Designing the Classified Ad](#)
- [Preserving Message Data at Run Time](#)
- [Creating the New Classified Ad Form](#)
- [Creating the Default Message](#)
- [Adding Custom Fields](#)
- [Setting the Message Class](#)

Designing the Classified Ad

The classified ad consists of several forms, each used for a different stage in the advertisement and sales process. Each form contains a set of custom properties that are useful to sellers and potential buyers. Also, each form is distinct, and rendered with its own ASP scripts, which are described briefly here:

1. **Advertisement Creation** The seller uses this form to create the initial sales (or "item wanted") announcement. When a form of this type is created, custom advertisement-specific fields such as Category, Subcategory, and ContactInfo are used.
2. **Advertisement Viewing** The potential buyer uses this form to read the sales announcement. This form renders the custom advertisement-specific fields and displays the values specified by the seller.

A different form is used to view the advertisement than to create it because certain controls are used only for one action and not the other. For example, the person submitting the ad first selects "Wanted" or "For Sale," because the item described in the ad may be either sought or offered. The viewer of the ad need only see the outcome of this choice – that the item is, for example, for sale. In the viewer's form, only one of "Wanted" or "For Sale" is displayed.

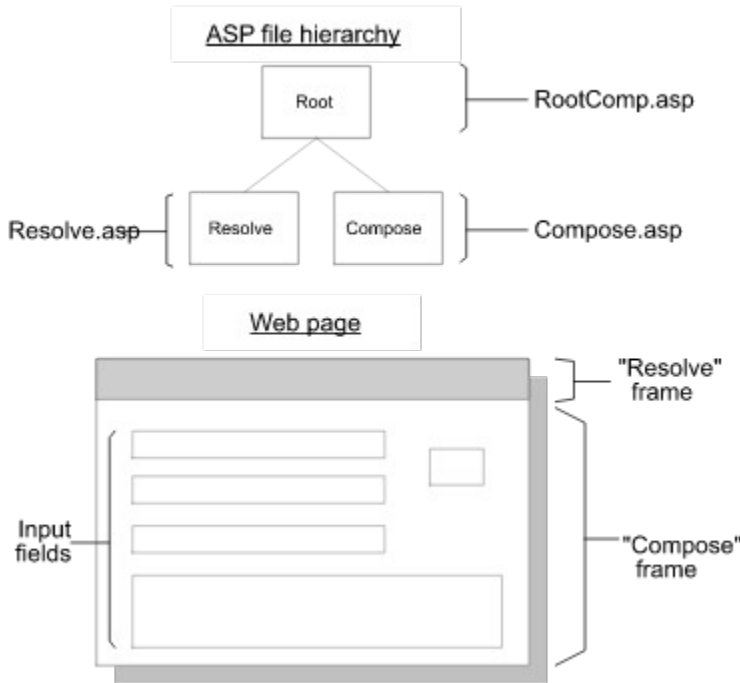
3. **Purchase Offer Creation** The buyer uses this form to respond to the advertisement.
4. **Purchase Offer Viewing** The seller uses this form to view the buyer's response.
5. **Purchase Offer Response** After the preceding forms are used to initiate a sale, one or more standard e-mail messages (of message class IPM.Note) can be sent to verify the details of the sale.

Preserving Message Data at Run Time

Because the World Wide Web uses pages instead of objects, special design considerations are required for Web forms. With Microsoft Exchange, if you send a message addressed to an invalid recipient, your message information remains intact in the message object, though the message is not delivered. You can readdress the message and send it again.

This model of data persistence cannot be transferred from a traditional messaging system to the Web. If a person types message information on a Web page and submits the page, that Web page disappears and a new page appears. This new page can be passed the user's information and can send it. But if this send attempt fails, perhaps because of an invalid recipient, the original page (along with the user's information) is gone. At this point, there is no convenient way to repopulate the fields with the user's message information.

The Classified Ad sample application has a solution to this problem that uses a three-part model. It uses the three script files `RootComp.asp`, `Resolve.asp`, and `Compose.asp`, as shown in the following diagram. (More frames are actually used than those shown here; this diagram is meant to illustrate merely how data is preserved.)



The files and frames in this illustration work together in the following manner.

At the Microsoft® Outlook™ Web Access interface, the user clicks a link that creates and renders a new classified ad. The first script called is `RootComp.asp`, which initializes and ties together two frames – `Compose` and `Resolve` – in one frameset. Since `Resolve` and `Compose` are two frames on the same page, both are running simultaneously.

The `Resolve` frame first checks a variable called `sendRequest`. If this variable has not been set to the value `sendMessage`, `Resolve` does not process the message. This mechanism is meant to keep `Resolve` from trying to process message information before the user has entered it. (At startup, VBScript automatically initializes the `sendRequest` variable to `Nothing`.)

`Resolve` displays a caption, but no tags for user input. Meanwhile, the `Compose` frame displays a form containing HTML controls with which the ad's creator enters information about offered or sought items. During this time, `Compose` sets the `sendRequest` variable to `sendMessage`.

When finished entering information, the user clicks **Send** on the `Compose` frame. Control is now

passed to Resolve, which, because *sendRequest* has been set to **sendMessage**, begins processing the message. Resolve requests the entered data from Compose, using the Request object in this manner:

```
szWorkPhone = Request.Form ("WorkPhone")
```

The Resolve frame now creates the new message object, adds fields for the data gathered from Compose, and sets their values. It also sets the form's type. For more information on these actions, see the following section.

Resolve checks essential fields and, if important data is missing or invalid (such as if the destination name does not resolve), it displays an alert to notify the user of the error. The user can now re-enter information at the Compose page, which still holds all the original information. If Resolve determines that the information is valid this time, it sends the message.

Creating the New Classified Ad Form

These are the main steps to follow for creating a form with CDO:

Group

To create a form using CDO

1. Using VBScript, create a message with the **Add** method of the Messages collection object. This call creates a standard message of the default message class IPM.Note, which contains the basic components needed to send a message, including the To, From, Received, and Subject fields and the message body.
2. You can now customize the standard message by adding custom fields to provide the special functionality the form offers.
3. Set the message class for the form and save the name of this message class in the **Type** property of the Message object.
4. If the form is to be used by Microsoft Outlook Web Access, save the form's .asp files to the appropriate location on disk.
5. If the form is to be used by the Microsoft Exchange Client – in addition to or instead of Web use – use the Microsoft Exchange Forms Manager to add the new form's message class to the Microsoft Exchange Server form library. For information on form libraries, see the *MAPI Programmer's Reference*.

These programming actions are discussed in more detail in the following sections.

Creating the Default Message

For the creation of a classified ad form, Read.asp is opened as the default file. In this script, the first step is the creation of a standard message. This happens in the following line of code, by the addition of a message to the current profile's (the session's) Outbox:

```
Set oNewMessage = objAMSession.Outbox.Messages.Add
```

In this call, *objAMSession* is the handle to the current session, and *oNewMessage* is the handle to the new message object. This new message contains the default fields of a standard message. The message handle is used later in this script when the message is sent with the command *oNewMessage.Send*. The **Send** command can include various flags, such as for requesting a delivery receipt or saving a copy in the Sent Items folder.

Because this call requests memory for a message object, it can fail if the computer is currently low on memory. For this reason, it may be good to check the return value from this call. In VBScript, if the object creation fails in a given call, the return value is set to **Nothing**. (In this code, rather than checking if the new message is set to **Nothing**, the creation of child objects is checked a few lines of code later.)

Adding Custom Fields

Within this new message object is a child object known as a Fields collection, which is the actual array of fields that house data. To start using the Fields collection, you may first want to obtain a handle to it with the following call:

```
Set oNewFields = oNewMessage.Fields
```

This call provides a handle to all the message's properties. After making this call, you can check its value for success (**Nothing** = failure). With the *oNewFields* handle, you can now add new fields to the message with the Fields.**Add** method, as in the following call:

```
oNewFields.Add "Category Name", Category_Type, Value
```

You can also determine the number of fields (or properties) in the message using Fields.**Count** (in this case, *oNewFields.Count*). For more information, see [Add Method \(Fields Collection\)](#). The code example at the bottom of the topic is reproduced here:

```
' 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
```

In this example, the message contains a Fields collection whose handle is *objFieldsColl*. Using the **Add** method, a new field called Keyword is created whose class is **vbString** and whose value is "Peru". Then, the variable *objNewField* is assigned ("Set") the value of this field creation operation, so that you can check for success. If the value assigned is not **Nothing**, the field was created.

Field Names and Field Identifiers

Some fields have names ("named properties"), and others do not, such as the FROM field. Unnamed fields are identified by their identifiers. It is easiest to use the named property when it exists, but you must use the property identifier otherwise. In the file *amprops.inc*, a number of custom property form name identifier numbers are defined. These are hexadecimal numbers used to identify fields. None of these fields are necessarily rendered, although they may contain data.

For example, in one sample script data is being retrieved from the FROM field. The Fields object is opened and the current index in the Fields object is set to the identifier of the FROM field. The identifier must be used because a given field (the FROM field in this example) is not always in the same position within a Fields collection. Because you can determine the number of properties in the Fields collection (with Fields.**Count**), you incrementally loop through the fields (from zero to the Fields count) until you reach the one whose identifier is equal to the constant defined for the FROM field. Then, you can read its value.

Setting the Message Class

After you have changed the set of fields, you need to provide a way for this form to be recognized (by a Microsoft Exchange client or by CDO) for the new, distinctive, type that it is. You do this by specifying and naming a new message class. You can give it any name you want, such as IPM.MyNote.

To set the message class, assign it to the **Type** property of the Message object. For example, this assignment occurs in Resolve.asp:

```
oNewMessage.Type = "IPM.Microsoft.ITG.NewClassifiedAd"
```

Before you make this assignment (for newly created objects), the Type field had a default setting of IPM.Note.

To have the Microsoft Exchange Client recognize a new message class, use the Microsoft Exchange Forms Manager to add it to the forms library. Then, you can view the new form through the **New Form** option on the **Compose** menu of the Microsoft Exchange Client.

Objects, Properties, and Methods

This reference contains property and method information for the Microsoft® Collaboration Data Objects (CDO) Library objects.

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

Object	Available since version	Properties	Methods
<u>AddressEntries</u> collection	1.1	Application, Class, Count, Filter, Item, Parent, RawTable1, Session	Add, Delete, GetFirst, GetLast, GetNext, GetPrevious, Sort
<u>AddressEntry</u>	1.0.a	Address, Application, Class, DisplayType, Fields, ID, Manager, MAPIOBJECT1, Members, Name, Parent, Session, Type	Delete, Details, GetFreeBusy, IsSameAs, Update
<u>AddressEntryFilter</u>	1.1	Address, Application, Class, Fields, Name, Not, Or, Parent, Session	IsSameAs
<u>AddressList</u>	1.1	AddressEntries, Application, Class, Fields, ID, Index, IsReadOnly, Name, Parent, Session	IsSameAs
<u>AddressLists</u> collection	1.1	Application, Class, Count, Item, Parent, Session	(none)
<u>AppointmentItem</u>	1.2	AllDayEvent, Application, Attachments, BusyStatus, Categories, Class, Conversation, ConversationIndex, ConversationTopic, DeliveryReceipt, Duration, Encrypted, EndTime, Fields, FolderID, ID, Importance,	ClearRecurrencePattern, CopyTo, Delete, GetRecurrencePattern, IsSameAs, MoveTo, Options, Respond, Send, Update

		IsRecurring, Location, MAPIOBJECT1, MeetingResponse Status, MeetingStatus, Organizer, Parent, ReadReceipt, Recipients, ReminderMinutes BeforeStart, ReminderSet, ReplyTime, ResponseRequest ed, Sender, Sensitivity, Sent, Session, Signed, Size, StartTime, StoreID, Subject, Submitted, Text, TimeCreated, TimeExpired, TimeLastModified, TimeReceived, TimeSent, Type, Unread	
<u>Attachment</u>	1.0.a	Application, Class, Fields, Index, MAPIOBJECT1, Name, Parent, Position, Session, Source, Type	Delete, IsSameAs, ReadFromFile, WriteToFile
<u>Attachments</u> collection	1.0.a	Application, Class, Count, Item, Parent, Session	Add, Delete
<u>Field</u>	1.0.a	Application, Class, ID, Index, Name, Parent, Session, Type, Value	Delete, ReadFromFile, WriteToFile
<u>Fields</u> collection	1.0.a	Application, Class, Count, Item, Parent, Session	Add, Delete, SetNamespace
<u>Folder</u>	1.0.a	Application, Class, Fields, FolderID, Folders, HiddenMessages, ID, MAPIOBJECT1, Messages, Name, Parent, Session,	CopyTo, Delete, IsSameAs, MoveTo, Update

		StoreID	
<u>Folders</u> collection	1.0.a	Application, Class, Count, Item, Parent, RawTable1, Session	Add, Delete, GetFirst, GetLast, GetNext, GetPrevious, Sort
<u>GroupHeader</u>	1.1	Application, Class, Count, Level, Name, Parent, Session, Unread	(none)
<u>InfoStore</u>	1.0.a	Application, Class, Fields, ID, Index, MAPIOBJECT1, Name, Parent, ProviderName, RootFolder, Session	IsSameAs
<u>InfoStores</u> collection	1.0.a	Application, Class, Count, Item, Parent, Session	(none)
<u>MeetingItem</u>	1.2	Application, Attachments, Categories, Class, Conversation, ConversationIndex, ConversationTopic, DeliveryReceipt, Encrypted, Fields, FolderID, ID, Importance, MAPIOBJECT1, MeetingType, Parent, ReadReceipt, Recipients, Sender, Sensitivity, Sent, Session, Signed, Size, StoreID, Subject, Submitted, Text, TimeCreated, TimeExpired, TimeLastModified, TimeReceived,	CopyTo, Delete, Forward, GetAssociatedAppointment, IsSameAs, MoveTo, Options, Reply, ReplyAll, Respond, Send, Update

		TimeSent, Type, Unread	
<u>Message</u>	1.0.a	Application, Attachments, Categories, Class, Conversation, ConversationIndex , ConversationTopic , DeliveryReceipt, Encrypted, Fields, FolderID, ID, Importance, MAPIOBJECT1, Parent, ReadReceipt, Recipients, Sender, Sensitivity, Sent, Session, Signed, Size, StoreID, Subject, Submitted, Text, TimeCreated, TimeExpired, TimeLastModified, TimeReceived, TimeSent, Type, Unread	CopyTo, Delete, Forward, IsSameAs, MoveTo, Options, Reply, ReplyAll, Send, Update
<u>MessageFilter</u>	1.1	Application, Class, Conversation, Fields, Importance, Not, Or, Parent, Recipients, Sender, Sent, Session, Size, Subject, Text, TimeFirst, TimeLast, Type, Unread	IsSameAs
<u>Messages</u> collection	1.0.a	Application, Class, Count, Filter, Item, Parent, RawTable1, Session	Add, Delete, GetFirst, GetLast, GetNext, GetPrevious, Sort
<u>Recipient</u>	1.0.a	Address, AddressEntry, AmbiguousNames, Application, Class, DisplayType, ID, Index, MeetingResponse	Delete, GetFreeBusy, IsSameAs, Resolve

		Status, Name, Parent, Session, Type	
<u>Recipients</u> collection	1.0.a	Application, Class, Count, Item, RawTable1, Parent, Resolved, Session	Add, AddMultiple, Delete, GetFirstUnresol ved, GetFreeBusy, GetNextUnresol ved, Resolve
<u>RecurrencePatter n</u>	1.2	Application, Class, DayOfMonth, DayOfWeekMask, Duration, EndTime, Instance, Interval, MonthOfYear, NoEndDate, Occurrences, Parent, PatternEndDate, PatternStartDate, RecurrenceType, Session, StartTime	(none)
<u>Session</u>	1.0.a	AddressLists, Application, Class, CurrentUser, Inbox, InfoStores, MAPIOBJECT1, Name, OperatingSystem, Outbox, OutOfOffice, OutOfOfficeText, Parent, Session, Version	AddressBook, CompareIDs, CreateConversatio nIndex, DeliverNow, GetAddressEntry, GetAddressList, GetArticle, GetDefaultFolder, GetFolder, GetInfoStore, GetMessage, GetOption, Logoff, Logon, SetLocaleIDs, SetOption

1 The **MAPIOBJECT** and **RawTable** properties are not available to Visual Basic applications. For more information, see the references for these properties.

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 CDO Library Objects describes the properties that have the same meaning for all CDO Library objects. These are:

- **Application**
- **Class**

- Parent
- Session

Object Model

The object model for the CDO Library is hierarchical. The following table shows the containment hierarchy. Each indented object is 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
 - Fields collection
 - Field
 - AddressEntries collection
 - AddressEntry
 - Fields collection
 - Field
 - AddressEntryFilter
 - Fields collection
 - Field
- Folder (Inbox or Outbox)
 - Fields collection
 - Field
 - Folders collection
 - Folder
 - Fields collection
 - Field
 - [Folders ... Folder ...]
 - Messages collection
 - AppointmentItem
 - RecurrencePattern
 - GroupHeader
 - MeetingItem
 - Message
 - Attachments collection
 - Attachment
 - Fields collection
 - Field
 - Fields collection
 - Field
 - Recipients collection
 - Recipient
 - AddressEntry
 - Fields collection
 - Field
 - MessageFilter
 - Fields collection
 - Field
 - InfoStores collection
 - InfoStore
 - Fields collection
 - Field
 - Folder [as expanded under Folders]

The notation "[Folders ... Folder ...]" signifies that any Folder object can contain a Folders collection of subfolders, and each subfolder can contain a Folders collection of more subfolders, nested to an arbitrary level.

Properties Common to All CDO Library Objects

All CDO Library objects expose 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 CDO 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 CDO Library object hierarchy and has no parent.

These common properties do not correspond to MAPI properties and cannot be rendered into HTML hypertext by the CDO Rendering Library.

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 CDO Library objects and that have the same meaning for all objects.

Properties

| Name | Type | Access |
|---------------------------|----------------|---------------|
| <u>Application</u> | String | Read-only |
| <u>Class</u> | Long | Read-only |
| <u>Parent</u> | Object | Read-only |
| <u>Session</u> | Session object | Read-only |

Application Property (All CDO Library Objects) Group

The **Application** property returns the name of the active application, namely the Microsoft® Collaboration Data Objects (CDO) Library. Read-only.

Syntax

object.Application

Data Type

String

Remarks

The **Application** property always contains the string "Collaboration Data Objects".

By always returning the same string, CDO 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. CDO, being part of the MAPI subsystem, is implemented with dynamic-link libraries, which take the extension .DLL.

Since version 1.1, CDO is an in-process server, residing in a .DLL file and linking dynamically with the calling modules. In comparison with the former local server architecture, this removes the need for remote procedure calls (RPCs) across process boundaries and greatly improves the performance of CDO Library calls.

The version number of the CDO Library is available through the [Session](#) object's **Version** property.

The **Application** property does not correspond to a MAPI property and cannot be rendered into HTML hypertext by the CDO Rendering Library.

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 ' or Dim objSession As MAPI.Session
' error handling ...
Set objSession = CreateObject("MAPI.Session")
If Not objSession Is Nothing Then
    objSession.Logon showDialog:=False
    MsgBox "Session's Application property = " & objSession.Application
End If
' error handling
End Function
```

Class Property (All CDO Library Objects)

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 CDO Library object. The following values are defined:

| CDO Library object | Class value | Type library constant |
|----------------------------------|-------------|--------------------------|
| <u>AddressEntries</u> collection | 21 | CdoAddressEntries |
| <u>AddressEntry</u> | 8 | CdoAddressEntry |
| <u>AddressEntryFilter</u> | 9 | CdoAddressFilter |
| <u>AddressList</u> | 7 | CdoAddressList |
| <u>AddressLists</u> collection | 20 | CdoAddressLists |
| <u>AppointmentItem</u> | 26 | CdoAppointment |
| <u>Attachment</u> | 5 | CdoAttachment |
| <u>Attachments</u> collection | 18 | CdoAttachments |
|
 | | |
| <u>Field</u> | 6 | CdoField |
| <u>Fields</u> collection | 19 | CdoFields |
| <u>Folder</u> | 2 | CdoFolder |
|
 | | |
| <u>Folders</u> collection | 15 | CdoFolders |
|
 | | |
|
 | | |
| <u>GroupHeader</u> | 25 | CdoGroupHeader |
| <u>InfoStore</u> | 1 | CdoInfoStore |
| <u>InfoStores</u> collection | 14 | CdoInfoStores |
|
 | | |
| <u>MeetingItem</u> | 27 | CdoMeetingItem |
| <u>Message</u> | 3 | CdoMsg |
| <u>MessageFilter</u> | 10 | CdoMessageFilter |
| <u>Messages</u> collection | 16 | CdoMessages |

| | | |
|------------------------------|----|-----------------------------|
| <u>Recipient</u> | 4 | CdoRecipient |
| <u>Recipients</u> collection | 17 | CdoRecipients |
| <u>RecurrencePattern</u> | 28 | CdoRecurrencePattern |
| <u>Session</u> | 0 | CdoSession |

CDO also defines **CdoUnknown**, with the value -1, for an object implementing the OLE **IUnknown** interface.

The **Class** property does not correspond to a MAPI property and cannot be rendered into HTML hypertext by the CDO Rendering Library.

Example

```
' Function: Util_DecodeObjectClass
' Purpose: Decode the long integer class value,
'          show the related object name
' See documentation topic: Class property
Function Util_DecodeObjectClass(IClass As Long)
' error handling here ...
Select Case (IClass)
    Case CdoSession:
        MsgBox ("Session object; Class = " & IClass)
    Case CdoMsg:
        MsgBox ("Message object; Class = " & IClass)
End Select
' error handling here ...
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 CdoSession = 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 CdoMessage = 3 for Message object
Util_DecodeObjectClass (objOneMsg.Class)
' error handling here ...
End Function
```

Parent Property (All CDO Library Objects) Group

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

Syntax

Set *objParent* = *object.Parent*

Data Type

Object

Remarks

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

| CDO 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 |
| AppointmentItem | Messages collection |
| Attachment | Attachments collection |
| Attachments collection | Message |
| Field | Fields collection |
| Fields collection | AddressEntry , AddressEntryFilter , AddressList , AppointmentItem , Attachment , Folder , InfoStore , Message , or MessageFilter |
| Folder (Inbox or Outbox) | Session |
| Folder (all others) | Folders collection or InfoStore |
| Folders collection | Folder , including Inbox or Outbox |
| GroupHeader | Messages collection |
| InfoStore | InfoStores collection |
| InfoStores collection | Session |
| MeetingItem | Messages collection |
| Message | Messages collection |
| MessageFilter | Messages collection |
| Messages collection | Folder , including Inbox or Outbox |
| Recipient | Recipients collection |

Recipients collection

Message

RecurrencePattern

AppointmentItem

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 CDO Library objects and its **Parent** property is set to **Nothing**.

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

The **Parent** property does not correspond to a MAPI property and cannot be rendered into HTML hypertext by the CDO Rendering Library. Depending on the parent object returned, you might be able to render it as an object, by setting the ObjectRenderer object's **DataSource** property to the object returned by the **Parent** property, or as a container object, by setting the ContainerRenderer object's **DataSource** property to the collection object returned by the **Parent** property. See the **DataSource** properties for what objects are accepted.

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 Messages collection available"
    Exit Function
End If
MsgBox "Messages collection parent class = " & objMessages.Parent.Class
Exit Function
' error handling here ...
End Function
```

Session Property (All CDO Library Objects) Group

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

Syntax

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

Data Type

Object (Session)

Remarks

The Session object represents the highest level in the CDO Library object hierarchy. If you invoke the **Session** property of a Session object, it returns the same Session object.

The **Session** property does not correspond to a MAPI property and cannot be rendered into HTML hypertext by the CDO Rendering Library.

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.

At a Glance

| | |
|----------------------------|--|
| Specified in type library: | CDO.DLL |
| First available in: | CDO 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** property has limited validity, and the best way to access an individual AddressEntry object within the collection is to use either its unique identifier or the **Get** methods. For more information on collections, see [Object Collections](#).

Properties

| Name | Available since 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 |
| RawTable | 1.1 | IUnknown object | Read/write
(Note: Not available to Visual Basic applications) |
| Session | 1.1 | Session object | Read-only |

Methods

| Name | Available since version | Parameters |
|--------------------------|-------------------------|--|
| Add | 1.1 | <i>emailtype</i> as String ,
(optional) <i>name</i> as String ,
(optional) <i>address</i> as String |
| Delete | 1.1 | (none) |
| GetFirst | 1.1 | (none) |
| GetLast | 1.1 | (none) |

| | | |
|---------------------------|-----|---|
| <u>GetNext</u> | 1.1 | (none) |
| <u>GetPrevious</u> | 1.1 | (none) |
| <u>Sort</u> | 1.1 | (optional) <i>SortOrder</i> as Long ,
(optional) <i>PropTag</i> as Long ,
(optional) <i>PropID</i> 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 container.

An AddressEntries collection can be rendered into HTML hypertext in tabular form using the CDO Rendering ContainerRenderer object. To specify this, set the container renderer's **DataSource** property to the AddressEntries collection object itself.

With the same **DataSource** setting, the container renderer's **RenderProperty** method can also render selected properties of the collection's parent AddressList object. The individual properties that can be rendered are indicated in the AddressList object property descriptions.

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**, **GetLast**, **GetNext**, 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 Microsoft® Visual Basic® **For Each** construction.

The order that items are returned by **GetFirst**, **GetLast**, **GetNext**, 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)

Group

The **Add** method creates and returns a new [AddressEntry](#) object in the AddressEntries collection.

Syntax

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

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.

You can set the *emailtype* parameter to any string recognized and supported by the address book providers invoked by the current profile, such as SMTP or X400. In particular, if you are using the Microsoft® Exchange private address book (PAB) provider, you can set *emailtype* to MAPIPDL to indicate a private distribution list (PDL).

The **DisplayType** property of the new AddressEntry object is set by the address book provider to either **CdoUser** or **CdoDistList**, depending on which kind of address entry is being added. The **DisplayType** property is read-only and cannot subsequently be changed.

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

The CDO Library does not support addition, deletion, or modification of members of a distribution list (DL) on the global address list (GAL).

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 the [AddressLists](#) collection.

```
' get PAB AddressList from AddressLists collection of Session
Dim myList As AddressList ' used to obtain PAB
Dim newEntry As AddressEntry ' new user in PAB
Dim newField As Field ' fax number of new user
' assume objSession created and logged on to without problem
Set myList = objSession.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  
Set newField = newEntry.Fields.Add("FaxNumber", vbString)  
newField.Value = "+1-206-555-7069" ' could have supplied this in Add  
' commit new entry, field, and value to PAB AddressList  
newEntry.Update
```

Count Property (AddressEntries Collection) Group

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

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 the value &H7FFFFFFF. Programmers needing to access individual objects in a large collection are strongly advised to use the Microsoft® 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, CDO returns &H7FFFFFFF (= $2^{31} - 1 = 2,147,483,647$) for the **Count** property. This is the largest positive value for a long integer and is intended to prevent an approximate count from prematurely terminating an indexed loop. On 32-bit platforms, this value is defined in the type library as **CdoMaxCount**. On other platforms, **CdoMaxCount** is not defined, and a program on such a platform must compare the **Count** property against &H7FFFFFFF to see if it is reliable.

If the **Count** property is not reliable, that is, if it is &H7FFFFFFF, a program using it to terminate an indexed loop must also check each returned object for a value of **Nothing** to avoid going past the end of the collection.

The personal address book (PAB) provider of Microsoft® Exchange does not provide a reliable count of its members, and the **Count** property returns **CdoMaxCount** whenever it is read from this provider.

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 indx As Long           ' loop index / object counter
Dim myPAB as AddressList   ' personal address book AddressList
Dim myPABColl as AddressEntries ' AddressEntries collection of PAB
Dim objOneAE as AddressEntry ' single address entry in collection
Dim objSess As MAPI.Session ' use early binding for efficiency
Set objSess = CreateObject ("MAPI.Session")
objSess.Logon showDialog:=False
' select PAB from AddressLists collection of Session
Set myPAB = objSess.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
indx = myPABColl.Count ' valid if not a "very large number"
If 0 = indx Then ' collection empty
    MsgBox "No AddressEntry items in PAB"
Elseif CdoMaxCount = indx Then ' .Count is not valid; get exact count
    indx = 0 ' set up to count individual address entries
    For Each objOneAE in myPABColl
        indx = indx + 1 ' another valid AddressEntry in collection
    Next objOneAE
End If
```

Delete Method (AddressEntries Collection) Group

The **Delete** method removes all the AddressEntry objects from the AddressEntries collection.

Syntax

objAddrEntriesColl.Delete()

Remarks

The **Delete** method performs an irreversible operation on the collection. It calls **Release** on the collection's reference to every AddressEntry object. If you have another reference to an address entry, you can still access its properties and methods, but you can never again associate it with any collection because the **Add** method always creates a new object. You should **Set** your reference variable either to **Nothing** or to another address entry.

The final **Release** on each AddressEntry object takes place when you assign your reference variable to **Nothing**, or when you call **Delete** if you had no other reference. At this point the object is removed from memory. Attempted access to a released object results in an error return of **CdoE_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 is still valid, and you can **Add** new members to it.

The CDO Library does not support addition, deletion, or modification of members of a distribution list (DL) on the global address list (GAL).

Filter Property (AddressEntries Collection) Group

The **Filter** property returns an [AddressEntryFilter](#) object for the AddressEntries collection. Read/write.

Syntax

objAddrEntriesColl.Filter

Data Type

Object (AddressEntryFilter)

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](#).

An address entry filter can also be inherited from the restriction specified in a CDO Rendering [TableView](#) object. Writing any property on this filter disinherits it, refreshes the AddressEntries collection, and instantiates a new address entry filter specifying only the property just written. This new filter, however, is no longer inherited, and the application can read its properties and set additional restrictions within it.

The address entry filter affects traversals of the AddressEntries collection using the Microsoft® Visual Basic® **For Each** statement, the **Get** methods, or the Visual Basic **For ... Next** construction. These accesses return an [AddressEntry](#) object.

Example

This code fragment shows how to set a filtering value in an AddressEntries collection's initial default address entry filter, and then how to clear all settings and reset the filter to its default state of no criteria:

```
Dim objAEColl As AddressEntries ' collection
Dim objAEntry As AddressEntry ' address entry passed by filter
Dim objAEFilt As AddressEntryFilter
' assume valid AddressEntries collection just created
' make first use of filter to check for names containing "Mac"
Set objAEFilt = objAEColl.Filter ' original empty default filter
objAEFilt.Name = "Mac" ' string used in a name resolution search
For Each objAEntry in objAEColl ' loops and Sets each objAEntry
    ' process address entries that are passed by the filter
Next
' ... later, when current filter settings are no longer needed ...
objAEColl.Filter = Nothing ' invalidates and clears filter
Set objAEFilt = objAEColl.Filter ' new empty filter
' filter now available for new settings
```

GetFirst Method (AddressEntries Collection) Group

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**()

objAddressEntry

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

objAddrEntriesColl

Required. The AddressEntries collection object.

Remarks

The order that items are returned by **GetFirst**, **GetLast**, **GetNext**, 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.

GetLast Method (AddressEntries Collection) Group

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**()

objAddressEntry

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

objAddrEntriesColl

Required. The AddressEntries collection object.

Remarks

The order that items are returned by **GetFirst**, **GetLast**, **GetNext**, 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.

If an AddressEntries collection has not been enumerated since it was initialized, the behavior of the **GetLast** method is not defined. A collection is enumerated when you call its **GetFirst** method, access a member through its **Item** property, or use its **Filter** property to specify a message filter. Some providers may return **CdoE_CALL_FAILED** if you call **GetLast** before the collection is enumerated.

GetNext Method (AddressEntries Collection) Group

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**()

objAddressEntry

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

objAddrEntriesColl

Required. The AddressEntries collection object.

Remarks

The order that items are returned by **GetFirst**, **GetLast**, **GetNext**, 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.

If an AddressEntries collection has not been enumerated since it was initialized, the behavior of the **GetNext** method is not defined. A collection is enumerated when you call its **GetFirst** method, access a member through its **Item** property, or use its **Filter** property to specify a message filter. Some providers may return **CdoE_CALL_FAILED** if you call **GetNext** before the collection is enumerated.

Calling **GetNext** on an unenumerated collection can produce unexpected results if the collection is reinitialized with a **Set** statement in every iteration of a loop. The recommended procedure is to set an explicit variable for the collection before entering the loop. For more information, see [Object Collections](#).

GetPrevious Method (AddressEntries Collection) Group

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**()

objAddressEntry

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

objAddrEntriesColl

Required. The AddressEntries collection object.

Remarks

The order that items are returned by [GetFirst](#), [GetLast](#), [GetNext](#), 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.

If an AddressEntries collection has not been enumerated since it was initialized, the behavior of the **GetPrevious** method is not defined. A collection is enumerated when you call its [GetFirst](#) method, access a member through its [Item](#) property, or use its [Filter](#) property to specify a message filter. Some providers may return **CdoE_CALL_FAILED** if you call **GetPrevious** before the collection is enumerated.

Calling **GetPrevious** on an unenumerated collection can produce unexpected results if the collection is reinitialized with a **Set** statement in every iteration of a loop. The recommended procedure is to set an explicit variable for the collection before entering the loop. For more information, see [Object Collections](#).

Item Property (AddressEntries Collection) Group

The **Item** property returns a single [AddressEntry](#) object from the AddressEntries collection. Read-only.

Syntax

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

objAddrEntriesColl.**Item**(*searchValue*)

index

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

searchValue

A string used to search the AddressEntries collection starting at the current position. The search returns the next AddressEntry object having the current sorting property greater than or equal to the *searchValue* string.

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

Data Type

Object (AddressEntry)

Remarks

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 returns the AddressEntry object at the indicated position in the collection. It can be used in an indexed loop, such as the **For ... Next** construction in Visual Basic. The first item in the collection has an index of 1.

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

The **Item**(*searchValue*) syntax returns the next AddressEntry object whose current sorting property is greater than or equal to the string specified by *searchValue*. This syntax starts its search at the current position.

Searching is based on the current sort order of the collection. The default sort property for an AddressEntries collection is the **Name** property of the collection's [AddressEntry](#) objects. If you want to use the **Item**(*searchValue*) syntax to search the collection on another property, for example an address type, you should first call the **Sort** method specifying the **Type** property.

Note The **Item**(*searchValue*) syntax uses the **IMAPITABLE::FindRow** method, which performs a search dependent on the current sort order of the table underlying the collection. Not all tables are sorted alphabetically. If your most recent sort order is nonalphabetic, you should access the messages using the **Item**(*index*) syntax. This could be the case, for example, if your AddressEntries collection is sorted on the **DisplayType** property.

For more information on tables, bookmarks, restrictions, and sort and search orders, see the *MAPI Programmer's Reference*.

Although the **Item** property itself is read-only, the [AddressEntry](#) object it returns can be accessed in the normal manner, and its properties retain their respective read/write or read-only accessibility.

RawTable Property (AddressEntries Collection) Group

The **RawTable** property returns an **IUnknown** pointer to the MAPI table object underlying the AddressEntries collection. Not available to Microsoft® Visual Basic® applications. Read/write.

Syntax

objAddrEntriesColl.**RawTable**

Data Type

Variant (**vbDataObject** format)

Remarks

The **RawTable** property is not available to Visual Basic programs. It is accessible only by C/C++ programs that deal with **IUnknown** objects. Visual Basic supports the **IDispatch** interface and not **IUnknown**. The **RawTable** property is an **IUnknown** object that returns an **IMAPITable** interface in response to **QueryInterface**. For more information, see [Introduction to Automation](#) and [How Programmable Objects Work](#). Also see the "COM and ActiveX Object Services" section of the Microsoft Platform SDK.

If your application uses any **MAPIOBJECT** or **RawTable** properties, it must **Release** them all before calling the [Session](#) object's **Logoff** method. Failure to do so can result in unexpected behavior.

The **RawTable** property does not correspond to a MAPI property and cannot be rendered into HTML hypertext by the CDO Rendering Library.

Sort Method (AddressEntries Collection)



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

Syntax

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

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

objAddrEntriesColl

Required. The AddressEntries collection object.

SortOrder

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

| Value | Numeric value | Description |
|----------------------|---------------|--------------------------|
| CdoNone | 0 | No sort |
| CdoAscending | 1 | Ascending sort (default) |
| CdoDescending | 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 **CdoPR_EMAIL_ADDRESS**.

name

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 *name* 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 **CdoPR_DISPLAY_NAME** is used for the sort.

Each call to **Sort** generates an entirely new sort order based on the specified property. No previous sort order is retained or nested.

If the underlying messaging system does not support the sort criteria specified, for example descending order or MAPI named properties, the **Sort** method returns **CdoE_TOO_COMPLEX**.

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.

At a Glance

| | |
|----------------------------|--|
| Specified in type library: | CDO.DLL |
| First available in: | CDO 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 since 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 |
| Manager | 1.1 | AddressEntry object | Read-only |
| MAPIOBJECT | 1.1 | IUnknown object | Read/write
(Note: Not available to Visual Basic applications) |
| Members | 1.1 | 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 since version | Parameters |
|---------------------------|--------------------------------|--|
| <u>Delete</u> | 1.0.a | (none) |
| <u>Details</u> | 1.0.a | (optional) <i>parentWindow</i> as Long |
| <u>GetFreeBusy</u> | 1.2 | <i>StartTime</i> as Variant ,
<i>EndTime</i> as Variant ,
<i>Interval</i> as Long |
| <u>IsSameAs</u> | 1.1 | (required) <i>objAddrEntry2</i> as Object |
| <u>Update</u> | 1.0.a | (optional) <i>makePermanent</i> as Boolean ,
(optional) <i>refreshObject</i> as Boolean |

Remarks

An AddressEntry object can be rendered into HTML hypertext using the CDO Rendering ObjectRenderer object. To specify this, set the object renderer's **DataSource** property to the AddressEntry object itself. The individual properties that can be rendered with the RenderProperty method are indicated in the AddressEntry object property descriptions.

Address Property (AddressEntry Object)

Group

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 **Address** property corresponds to the MAPI property PR_EMAIL_ADDRESS. It can be rendered into HTML hypertext using the CDO Rendering ObjectRenderer object. To specify this, set the object renderer's **DataSource** property to this AddressEntry object and the *property* parameter of the **RenderProperty** method to **CdoPR_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)

Group

The **Delete** method removes the AddressEntry object from the [AddressEntries](#) collection.

Syntax

objAddressEntry.Delete()

Remarks

The **Delete** method performs an irreversible operation on the collection. It calls **Release** on the collection's reference to the AddressEntry object. If you have another reference to the address entry, you can still access its properties and methods, but you can never again associate it with any collection because the **Add** method always creates a new object. You should **Set** your reference variable either to **Nothing** or to another address entry.

The final **Release** on the AddressEntry object takes place when you assign your reference variable to **Nothing**, or when you call **Delete** if you had no other reference. At this point the object is removed from memory. Attempted access to a released object results in an error return of **CdoE_INVALID_OBJECT**.

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

When you delete a member of a collection, the collection is immediately refreshed, meaning that its **Count** property is reduced by one and its members are reindexed. To access a member following the deleted member, you must use its new index value. For more information, see [Looping Through a Collection](#).

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

The CDO Library does not support addition, deletion, or modification of members of a distribution list (DL) on the global address list (GAL).

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

This code fragment illustrates the two situations previously explained. The **Set** statement calls **AddRef** on the first AddressEntry object. That reference survives the call to **Delete** and has to be reassigned. The second AddressEntry object is deleted without creating another reference, and no other action is necessary.

```
' assume valid AddressList object
Set objAddressEntry = objAddressList.AddressEntries.Item(1)
objAddressEntry.Delete ' still have a reference from Set statement
' ... other operations on objAddressEntry possible but pointless ...
Set objAddressEntry = Nothing ' necessary to remove reference
' ...
objAddressList.AddressEntries.Item(2).Delete ' no reference to remove
```

Details Method (AddressEntry Object)

Group

The **Details** method displays a modal dialog box that provides detailed information about an AddressEntry object.

Syntax

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

objAddressEntry

Required. The AddressEntry object.

parentWindow

Optional. Long. The parent window handle for the details dialog box. A value of zero (the default) specifies that the dialog box should be application-modal.

Remarks

The **Details** dialog box is always modal, meaning the parent window is disabled while the dialog box 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 box is active. If the *parentWindow* parameter is supplied but is not valid, the call returns **CdoE_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 dialog boxes:

- **Details** method (AddressEntry object)
- **Options** and **Send** methods (Message object)
- **Resolve** method (Recipient object)
- **Resolve** method (Recipients collection)
- **AddressBook** and **Logon** methods (Session object)

However, if your application is running as a Microsoft® Windows NT® service, for example from Active Server Pages (ASP) script on a Microsoft® Internet Information Server (IIS), no user interface is allowed.

For more information on Windows NT services, see the Win32® Web page *Using MAPI from a Windows NT Service* at <http://www.microsoft.com/win32dev/mapi/mapiserv.htm>. For more information on running as a service, see "Windows NT Service Client Applications" in the *MAPI Programmer's Reference*.

DisplayType Property (AddressEntry Object) Group

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 | Decimal value | Description |
|---------------------------|---------------|---|
| CdoAgent | 3 | An automated agent, such as Quote-of-the-Day. |
| CdoDistList | 1 | A public distribution list. |
| CdoForum | 2 | A forum, such as a bulletin board or a public folder. |
| CdoOrganization | 4 | A special address entry defined for large groups, such as a helpdesk. |
| CdoPrivateDistList | 5 | A private, personally administered distribution list. |
| CdoRemoteUser | 6 | A messaging user in a remote messaging system. |
| CdoUser | 0 | A local messaging user. |

When you **Add** a new address entry to an [AddressEntries](#) collection, the **DisplayType** property is set by the address book provider to either **CdoUser** or **CdoDistList**, depending on which kind of address entry is being added. The **DisplayType** property cannot subsequently be changed.

If an address entry represents a distribution list, the **Members** property can be used to retrieve an [AddressEntries](#) collection containing the members of the distribution list. If the address entry is a single messaging user, the **Members** property returns **Nothing**.

A private distribution list (PDL) exists only in your personal address book (PAB) and does not have an e-mail address. Before invoking an address entry's **Address** or **Type** property, you should verify that its **DisplayType** is not **CdoPrivateDistList**. Attempted access to addressing properties on a PDL results in a return of **CdoE_NOT_FOUND**.

The **DisplayType** property corresponds to the MAPI property PR_DISPLAY_TYPE. It can be rendered into HTML hypertext using the CDO Rendering [ObjectRenderer](#) object. To specify this, set the object renderer's **DataSource** property to this AddressEntry object and the *property* parameter of the **RenderProperty** method to **CdoPR_DISPLAY_TYPE**.

Fields Property (AddressEntry Object)

Group

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

Integer. Value must be between 1 and 65535 (&HFFFF) inclusive. Specifies the index within the Fields collection.

proptag

Long. Value must be greater than or equal to 65,536 (&H10000). 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 (Field or Fields collection)

Remarks

The **Fields** property returns one or all of the fields associated with an AddressEntry object. Each field typically corresponds to a MAPI property. Data types are preserved, except that MAPI counted binary properties are converted to and from character strings representing hexadecimal digits.

The **Fields** property provides a generic access mechanism that allows Microsoft® Visual Basic® and Microsoft® 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 **CdoPR_GIVEN_NAME**. To access a named property, use *objAddressEntry.Fields(name)*, where *name* is a string that represents the custom property name.

Although the **Fields** property itself is read-only, the collection it returns can be accessed in the normal manner through its **Add** and **Delete** methods, and the properties on its member [Field](#) objects retain their respective read/write or read-only accessibility.

You must have the appropriate permissions to add, delete, or modify fields on an AddressEntry object. Normally you only have these permissions on your personal address book (PAB). Furthermore, some address book containers such as the global address list (GAL) support only a fixed set of properties. Even if you have permissions on a container with a fixed schema, you cannot add new fields to its entries. An attempt to do so returns **CdoE_NO_ACCESS**.

The **Fields** property does not correspond to a MAPI property and cannot be rendered into HTML hypertext by the CDO Rendering Library.

GetFreeBusy Method (AddressEntry Object) Group

The **GetFreeBusy** method returns a string representing the availability of the messaging user for a meeting over a specified period of time.

Syntax

strAvail = *objAddressEntry*.**GetFreeBusy**(*StartTime*, *EndTime*, *Interval*)

strAvail

On successful return, contains a string indicating the messaging user's availability for each of the time slots in the specified time period.

objAddressEntry

Required. The AddressEntry object.

StartTime

Required. Variant (**vbDate** format). Specifies the date/time of the beginning of the first time slot.

EndTime

Required. Variant (**vbDate** format). Specifies the date/time of the end of the last time slot.

Interval

Required. Long. Specifies the length of each time slot in minutes. If this parameter is less than 1, **GetFreeBusy** returns **CdoE_INVALID_PARAMETER**.

Remarks

The returned string length equals the number of time slots between *StartTime* and *EndTime*. Each character is the ASCII representation of the appropriate type library constant indicating the messaging user's availability during a time slot:

| ASCII character | Corresponding type library constant | Meaning |
|-----------------|-------------------------------------|--|
| "0" | CdoFree | Available for appointments or meetings throughout the time slot |
| "1" | CdoTentative | At least one tentative commitment during the time slot |
| "2" | CdoBusy | At least one confirmed commitment during the time slot |
| "3" | CdoOutOfOffice | Designated as out-of-office (OOO) for at least part of the time slot |

If there is any overlapping of commitments during a time slot, **GetFreeBusy** returns the most committed state, that is, the highest character value. For example, if a messaging user already has one tentative meeting and one confirmed meeting scheduled during the same time slot, **GetFreeBusy** returns "2" for that time slot, corresponding to **CdoBusy**. **CdoFree** is not returned unless the entire time slot is free of commitments.

For performance reasons, calendaring clients typically do not publish appointments indefinitely into the future. For example, the CDO Library, Microsoft® Outlook™, Microsoft® Schedule+, and Microsoft Outlook Web Access (OWA) all publish appointments for a default maximum of three months past the current date. This means that if you call **GetFreeBusy** on a messaging user and specify time slots more than three months in advance of that user's current date, **GetFreeBusy** is likely to return **CdoFree** for every such time slot. Unless you are familiar with the behavior and publishing limits of a messaging user's calendaring client, you should not interpret **CdoFree** as meaning that the user is

genuinely free for a time slot significantly in the future.

If an address entry represents a distribution list, the status of its individual members cannot be returned to you. A meeting request should be sent only to single messaging users. You can determine if a messaging user is a distribution list by checking its **DisplayType** property.

ID Property (AddressEntry Object)

Group

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

Syntax

objAddressEntry.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. However, MAPI does not require identifier values to be binary comparable. Accordingly, two identifier values can be different, yet refer to the same object. MAPI compares identifiers with the **CompareEntryIDs** method. CDO provides the **CompareIDs** method in the Session object. For more information on entry identifiers, see the *MAPI Programmer's Reference*.

Although the AddressEntry and Recipient objects are not identical objects in the CDO Library, they represent the same underlying MAPI messaging user object, and the address entry's **ID** property is equal to the recipient's **ID** property. This can be used to advantage, for example, when adding an existing AddressEntry object to a Recipients collection. You can use the address entry's **ID** property as the *entryID* parameter to the **Add** method.

The **ID** property corresponds to the MAPI property PR_ENTRYID, converted to a string of hexadecimal characters. It can be rendered into HTML hypertext using the CDO Rendering ObjectRenderer object. To specify this, set the object renderer's **DataSource** property to this AddressEntry object and the *property* parameter of the **RenderProperty** method to **CdoPR_ENTRYID**.

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 CDO 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) Group

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

Syntax

```
boolSame = objAddressEntry.IsSameAs(objAddrEntry2)
```

boolSame

On successful return, contains **True** if the two objects are the same.

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 in the underlying messaging system. 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. This is necessary because, although MAPI requires all entry identifiers to be unique, it does not require two of them identifying the same object to be identical. A generic comparison of any two objects' unique identifiers is available with the Session object's [CompareIDs](#) method.

Example

This code fragment uses **IsSameAs** to verify that the sender of a message is the same messaging user as is found in the receiver's personal address book (PAB):

```
Dim objMessage As Message
Dim objSender As AddressEntry
Dim colAEs As AddressEntries
Dim objAEFilt As AddressEntryFilter
Dim objUser As AddressEntry
' assume objMessage received without error
Set objSender = objMessage.Sender
' construct sender's full address
strSender = objSender.Type & ":" & objSender.Address
' get the Personal Address Book the easy way (1 = CdoAddressListPAB)
Set colAEs = objSession.GetAddressList(1).AddressEntries
Set objAEFilt = colAEs.Filter
objAEFilt.Address = strSender ' look for sender's full address
For Each objUser in colAEs
    If objUser.IsSameAs(objSender) Then
        MsgBox "Sender found in PAB"
    End If
Next
```

Manager Property (AddressEntry Object) Group

The **Manager** property returns an AddressEntry object representing the manager of the user corresponding to this address entry. Read-only.

Syntax

Set *objAddrEntry* = *objAddressEntry*.**Manager**

objAddrEntry

Object. The returned AddressEntry object that represents the manager of the messaging user represented by this address entry.

objAddressEntry

Object. This AddressEntry object.

Data Type

Object (AddressEntry)

Remarks

You can use the **Manager** property when your organization stores management information in the MAPI system. This is possible, for example, with Microsoft® Exchange Server.

If the user's manager is not available in the MAPI system, the **Manager** property returns **Nothing**.

The **Manager** property does not correspond to a MAPI property and cannot be rendered into HTML hypertext by the CDO Rendering Library. It could be rendered as an object by setting the ObjectRenderer object's **DataSource** property to the AddressEntry object returned by the **Manager** property.

The Name property of the AddressEntry object returned by the **Manager** property corresponds to the MAPI property PR_MANAGER_NAME. To render just the manager's name, you can set the object renderer's **DataSource** property to this AddressEntry object and the *property* parameter of the RenderProperty method to **CdoPR_MANAGER_NAME**.

MAPIOBJECT Property (AddressEntry Object) Group

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

Syntax

objAddressEntry.**MAPIOBJECT**

Data Type

Variant (**vbDataObject** format)

Remarks

The **MAPIOBJECT** property is not available to Visual Basic programs. It is accessible only by C/C++ programs that deal with **IUnknown** objects. Visual Basic supports the **IDispatch** interface and not **IUnknown**. The **MAPIOBJECT** property is an **IUnknown** object that returns an **IMailUser** interface in response to **QueryInterface**. For more information, see [Introduction to Automation](#) and [How Programmable Objects Work](#). Also see the "COM and ActiveX Object Services" section of the Microsoft Platform SDK.

If your application uses any **MAPIOBJECT** or **RawTable** properties, it must **Release** them all before calling the [Session](#) object's **Logoff** method. Failure to do so can result in unexpected behavior.

The **MAPIOBJECT** property does not correspond to a MAPI property and cannot be rendered into HTML hypertext by the CDO Rendering Library.

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 (AddressEntries collection)

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**. The address entry is a distribution list if its DisplayType property is set to **CdoDistList** or **CdoPrivateDistList**.

Although the **Members** property itself is read-only, the collection it returns can be accessed in the normal manner through its Add and Delete methods, and the properties on its member AddressEntry objects retain their respective read/write or read-only accessibility.

The **Members** property does not correspond to a MAPI property and cannot be rendered into HTML hypertext by the CDO Rendering Library. It could be rendered as a container object by setting the ContainerRenderer object's DataSource property to the AddressEntries collection object returned by the **Members** property.

Name Property (AddressEntry Object)

Group

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

Syntax

objAddressEntry.Name

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

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. It can be rendered into HTML hypertext using the ObjectRenderer object. To specify this, set the object renderer's **DataSource** property to this AddressEntry object and the *property* parameter of the **RenderProperty** method to **CdoPR_DISPLAY_NAME**.

Example

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

See Also

[Using Addresses](#)

Type Property (AddressEntry Object)

Group

The **Type** property specifies the address type, such as SMTP, FAX, or X400. 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 value of the **Type** property is not checked by CDO, but it should contain a string recognized and supported by the address book providers invoked by the current profile. For example, the Microsoft® Exchange private address book (PAB) provider supports a **Type** value of MAIPDL to denote a private distribution list (PDL).

The **Type** property corresponds to the MAPI property PR_ADDRTYPE. It can be rendered into HTML hypertext using the CDO Rendering ObjectRenderer object. To specify this, set the object renderer's **DataSource** property to this AddressEntry object and the *property* parameter of the RenderProperty method to **CdoPR_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*])

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 container. **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 container. **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, CDO 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. |

Calling **Update(False, True)** flushes the cache and then reloads the appropriate values from the address book. This effectively cancels any changes that had been made to the AddressEntry object in memory.

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

Example

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

```
' 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.

At a Glance

| | |
|----------------------------|---|
| Specified in type library: | CDO.DLL |
| First available in: | CDO Library version 1.1 |
| Parent objects: | AddressEntries collection |
| Child objects: | Fields collection |
| Default property: | Name |

Properties

| Name | Available since 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 since version | Parameters |
|--------------------------|-------------------------|---|
| IsSameAs | 1.1 | <i>objAddrEntryFilter2</i> 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 or adding fields to its Fields collection. You do not need to call any **Update** method when setting filter criteria.

The filter is invoked when the AddressEntries collection is traversed with the **Get** methods or the Microsoft® Visual Basic® **For Each** construction. Each field participates in a MAPI search restriction comparing the field's **Value** property against the value of the AddressEntry object's 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. However, the following MAPI properties are compared using FL_PREFIX instead of FL_SUBSTRING:

PR_ACCOUNT

PR_BUSINESS_ADDRESS_CITY
PR_COMPANY_NAME
PR_DEPARTMENT_NAME
PR_DISPLAY_NAME
PR_GIVEN_NAME
PR_OFFICE_LOCATION
PR_SURNAME
PR_TITLE

If the underlying messaging system does not support the search criteria specified by the filter fields, the **Get** methods return **CdoE_TOO_COMPLEX**. In particular, a given field may not be supported by a particular address book container.

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

The address entry filter affects traversals of the AddressEntries collection using the Visual Basic **For Each** statement, the **Get** methods, or the Visual Basic **For ... Next** construction. These accesses return an AddressEntry.

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. You can use the following code to clear an address entry filter of all of its previous settings and reset it to its default state of no criteria:

```
objAddrEntsColl.Filter = Nothing ' filter now invalidated and cleared  
Set objAddrEntFilt = objAddrEntsColl.Filter ' new valid empty filter
```

If an address book container is being rendered with a CDO rendering application, the AddressEntries collection and the address entry filter are instantiated according to the specifications in the TableView object being applied to the address book container. The AddressEntryFilter object inherits the restriction specified in the view. An inherited filter can be used without modification, but it cannot be read or changed by the rendering application. Writing any property on an inherited filter disinherits it and refreshes the AddressEntries collection. This means that the collection is reinstantiated with a new address entry filter specifying only the property just written. This new filter, however, is no longer inherited, and the application can read its properties and set additional restrictions within it.

Example

This code fragment specifies that the address entry filter on a personal address book (PAB) should pass only AddressEntry objects that represent remote users **OR** that have a fax address:

```
Dim objThisPAB As AddressList          ' Personal address book  
Dim objAddrEntFilt As AddressEntryFilter  
Dim propTag As Long                   ' MAPI property tag  
Dim dispType As Long                   ' MAPI display type  
  
Set objThisPAB = objSession.AddressLists("Personal Address Book")  
' ... validate AddressList object ...  
Set objAddrEntFilt = objThisPAB.AddressEntries.Filter  
' ... validate AddressEntryFilter object ...  
dispTypeTag = &H39000003 ' VB4.0: dispTypeTag = CdoPR_DISPLAY_TYPE  
dispTypeVal = 6          ' VB4.0: dispTypeVal = CdoRemoteUser  
objAddrEntFilt.Fields.Add(dispTypeTag, dispTypeVal)  
objAddrEntFilt.Address = "fax:" ' case-insensitive substring match
```

objAddrEntFilt.Or = True ' OR results together instead of AND

Address Property (AddressEntryFilter Object) Group

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 CDO from its Name property and the Resolve method.

Fields Property (AddressEntryFilter Object) Group

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

Integer. Value must be greater than 0 and less than or equal to 65,535 (&HFFFF). Specifies the index within the Fields collection.

proptag

Long. Value must be greater than or equal to 65,536 (&H10000). 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 (Field or Fields collection)

Remarks

The **Fields** property returns one or all of the fields associated with an AddressEntryFilter object. Each field typically corresponds to a MAPI property. Data types are preserved, except that MAPI counted binary properties are converted to and from character strings representing hexadecimal digits.

The fields that have been set in the [Fields](#) collection specify the filter, together with any other AddressEntryFilter properties that have been set.

The **Fields** property provides a generic access mechanism that allows Microsoft® Visual Basic® and Microsoft® 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 **CdoPR_POSTAL_ADDRESS**. To access a named property, use *objAddressEntryFilter.Fields(name)*, where *name* is a string that represents the custom property name.

Note Not all address book containers support filtering on all possible fields, that is, on all possible MAPI properties. For example, the Microsoft® Exchange provider for the global address list (GAL) only supports filtering on **CdoPR_DISPLAY_NAME**, and returns **CdoE_TOO_COMPLEX** if you specify any other property.

Although the **Fields** property itself is read-only, the collection it returns can be accessed in the normal manner through its **Add** and **Delete** methods, and the properties on its member [Field](#) objects retain their respective read/write or read-only accessibility.

IsSameAs Method (AddressEntryFilter object) Group

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

Syntax

boolSame = *objAddressEntryFilter*.**IsSameAs**(*objAddrEntryFilter2*)

boolSame

On successful return, contains **True** if the two objects are the same.

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 in the underlying messaging system. 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. This is necessary because, although MAPI requires all entry identifiers to be unique, it does not require two of them identifying the same object to be identical. A generic comparison of any two objects' unique identifiers is also available with the Session object's **CompareIDs** method.

Name Property (AddressEntryFilter Object) Group

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

Syntax

objAddressEntryFilter.Name

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

Data Type

String

Remarks

The **Name** property contains an 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.

Example

This code fragment specifies that the address entry filter should pass all AddressEntry objects that contain any resolution of the string "pet", such as "Peter", "Petra", "Peterson", and "pet lovers":

```
Dim colAEs As AddressEntries
Dim objAEFilt As AddressEntryFilter
Dim objAEpet As AddressEntry
' ... validate AddressEntries collection ...
Set objAEFilt = colAEs.Filter
' ... validate AddressEntry object ...
objAEFilt.Name = "pet"
For Each objAEpet in colAEs
    MsgBox "Found " & objAEpet ' .Name is default property
Next
```

Not Property (AddressEntryFilter Object) Group

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)

Group

The **Or** property specifies that the restriction values are to be **ORed** instead of **ANDed** 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 **ANDed** together. If it is **True**, the values are **ORed** together.

AddressList Object

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

At a Glance

| | |
|----------------------------|--|
| Specified in type library: | CDO.DLL |
| First available in: | CDO Library version 1.1 |
| Parent objects: | AddressLists collection |
| Child objects: | AddressEntries collection
Fields collection |
| Default property: | Name |

Properties

| Name | Available since version | Type | Access |
|---------------------------------------|-------------------------|--|-----------|
| AddressEntries | 1.1 | AddressEntries collection object | Read-only |
| Application | 1.1 | String | Read-only |
| Class | 1.1 | Long | Read-only |
| Fields | 1.1 | Field object or Fields collection object | 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 since version | Parameters |
|---------------------------------|-------------------------|--------------------------------------|
| IsSameAs | 1.1 | <i>objAddrList2</i> 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.

An AddressList object can be rendered into HTML hypertext as the parent of an [AddressEntries](#) collection, using the [ContainerRenderer](#) object. The individual properties that can be rendered with the [**RenderProperty**](#) method are indicated in the AddressList object property descriptions.

AddressEntries Property (AddressList Object) Group

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 (AddressEntry or AddressEntries collection)

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.

Although the **AddressEntries** property itself is read-only, the collection it returns can be accessed in the normal manner through its **Add** and **Delete** methods, and the properties on its member [AddressEntry](#) objects retain their respective read/write or read-only accessibility.

The **AddressEntries** property does not correspond to a MAPI property and cannot be rendered into HTML hypertext by the CDO Rendering Library. If a single [AddressEntry](#) object is returned, it could be rendered as an object by setting the [ObjectRenderer](#) object's **DataSource** property to the AddressEntry object returned by the **AddressEntries** property. If an [AddressEntries](#) collection is returned, it could be rendered as a container object by setting the [ContainerRenderer](#) object's **DataSource** property to the AddressEntries collection object returned by the **AddressEntries** property.

Fields Property (AddressList Object)

Group

The **Fields** property returns a single [Field](#) object or a [Fields](#) collection object. Read-only.

Syntax

objAddressList.Fields

objAddressList.Fields(*index*)

objAddressList.Fields(*proptag*)

objAddressList.Fields(*name*)

index

Integer. Value must be greater than 0 and less than or equal to 65,535 (&HFFFF). Specifies the index within the Fields collection.

proptag

Long. Value must be greater than or equal to 65,536 (&H10000). 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 (Field or Fields collection)

Remarks

The **Fields** property returns one or all of the fields associated with an AddressList object. Each field typically corresponds to a MAPI property. Data types are preserved, except that MAPI counted binary properties are converted to and from character strings representing hexadecimal digits.

The **Fields** property provides a generic access mechanism that allows Microsoft® Visual Basic® and Microsoft® 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 *objAddressList.Fields(proptag)*, where *proptag* is the 32-bit MAPI property tag associated with the property, such as **CdoPR_DISPLAY_NAME**. To access a named property, use *objAddressList.Fields(name)*, where *name* is a string that represents the custom property name.

Although the **Fields** property itself is read-only, the collection it returns can be accessed in the normal manner through its **Add** and **Delete** methods, and the properties on its member [Field](#) objects retain their respective read/write or read-only accessibility.

The **Fields** property does not correspond to a MAPI property and cannot be rendered into HTML hypertext by the CDO Rendering Library.

ID Property (AddressList Object)

Group

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. However, MAPI does not require identifier values to be binary comparable. Accordingly, two identifier values can be different, yet refer to the same object. MAPI compares identifiers with the **CompareEntryIDs** method. CDO provides the **CompareIDs** method in the Session object. For more information on entry identifiers, see the *MAPI Programmer's Reference*.

The **ID** property corresponds to the MAPI property PR_ENTRYID, converted to a string of hexadecimal characters. It can be rendered into HTML hypertext using the CDO Rendering ContainerRenderer object if the container renderer's **DataSource** property is set to this AddressList object's child AddressEntries collection. To specify this, set the *property* parameter of the **RenderProperty** method to **CdoPR_ENTRYID**.

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)

Group

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. It can be saved and used later with the collection's **Item** property to reselect the same address list in the collection.

The first object in the collection has an **Index** value of 1.

The **Index** property does not correspond to a MAPI property and cannot be rendered into HTML hypertext by the CDO Rendering Library.

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 ' requested address list  
' 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  
' get rqIndex by passed parameter or by prompting ...  
Set objOneAddressList = objAddressListsColl.Item(rqIndex)  
If objOneAddressList Is Nothing Then  
    MsgBox "AddressList could not be selected"  
    Exit Function  
End If  
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) Group

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.

The **IsReadOnly** property does not correspond to a MAPI property and cannot be rendered into HTML hypertext by the CDO Rendering Library.

IsSameAs Method (AddressList Object)

Group

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

Syntax

```
boolSame = objAddressList.IsSameAs(objAddrList2)
```

boolSame

On successful return, contains **True** if the two objects are the same.

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 in the underlying messaging system. 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. This is necessary because, although MAPI requires all entry identifiers to be unique, it does not require two of them identifying the same object to be identical. A generic comparison of any two objects' unique identifiers is also available with the Session object's **CompareIDs** method.

Name Property (AddressList Object)

Group

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

Syntax

objAddressList.Name

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

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. It can be rendered into HTML hypertext using the CDO Rendering ContainerRenderer object if the container renderer's **DataSource** property is set to this AddressList object's child AddressEntries collection. To specify this, set the *property* parameter of the **RenderProperty** method to **CdoPR_DISPLAY_NAME**.

Example

```
Dim objAddressList As AddressList ' assume valid address list object
MsgBox "Address book container name = " & objAddressList.Name
' or could be just objAddressList since .Name is default property
```

AddressLists Collection Object

The AddressLists collection object contains one or more AddressList objects.

At a Glance

| | |
|----------------------------|-----------------------------|
| Specified in type library: | CDO.DLL |
| First available in: | CDO 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 Microsoft® Visual Basic® **For Each** statement. For more information on collections, see [Object Collections](#).

Properties

| Name | Available since 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) Group

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 display the name of every valid AddressList object in the collection:

```
Dim i As Integer ' loop counter
Dim hierarchy As AddressLists ' address book hierarchy
' assume valid session and successful logon
Set hierarchy = objSession.AddressLists
' make sure collection 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 lists in hierarchy
If 0 = i Then
    MsgBox "No available address books"
    ' exit ...
End If
' display names of all valid address book containers
For i = 1 To hierarchy.Count Step 1
    If Nothing = hierarchy.Item(i) Then
        MsgBox "Address List " & i & " is not valid"
    Else
        MsgBox "Address List " & i & ": " & hierarchy(i).Name
    End If
Next i
```

Item Property (AddressLists Collection)

Group

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.

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

Data Type

Object (AddressList)

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*.

Although the **Item** property itself is read-only, the AddressList object it returns can be accessed in the normal manner, and its properties retain their respective read/write or read-only accessibility.

AppointmentItem Object

The AppointmentItem object represents an appointment in a calendar folder.

At a Glance

| | |
|----------------------------|---|
| Specified in type library: | CDO.DLL |
| First available in: | CDO Library version 1.2 |
| Parent objects: | Messages collection |
| Child objects: | Attachments collection
Fields collection
Recipients collection
RecurrencePattern |
| Default property: | Subject |

The AppointmentItem object is a subclass of the [Message](#) object and exposes all the same properties and methods, except for the Message object's [Forward](#), [Reply](#), and [ReplyAll](#) methods. In the following tables of properties and methods, those that are common with the Message object are linked to their descriptions for the Message object. Only the properties and methods unique to the AppointmentItem object are described in this section.

Properties

| Name | Available since version | Type | Access |
|-----------------------------------|-------------------------|--|------------|
| AllDayEvent | 1.2 | Boolean | Read/write |
| Application | 1.2 | String | Read-only |
| Attachments | 1.2 | Attachment object or Attachments collection object | Read-only |
| BusyStatus | 1.2 | Long | Read/write |
| Categories | 1.2 | String array | Read/write |
| Class | 1.2 | Long | Read-only |
| Conversation | 1.2 | (Obsolete. Do not use.) | Read/write |
| ConversationIndex | 1.2 | String | Read/write |
| ConversationTopic | 1.2 | String | Read/write |
| DeliveryReceipt | 1.2 | Boolean | Read/write |
| Duration | 1.2 | Long | Read-only |
| Encrypted | 1.2 | Boolean | Read/write |
| EndTime | 1.2 | Variant (vbDate format) | Read/write |
| Fields | 1.2 | Field object or Fields collection object | Read-only |
| FolderID | 1.2 | String | Read-only |
| ID | 1.2 | String | Read-only |
| Importance | 1.2 | Long | Read/write |

| | | | |
|--|-----|---|--|
| <u>IsRecurring</u> | 1.2 | Boolean | Read-only |
| <u>Location</u> | 1.2 | String | Read/write |
| <u>MAPIOBJECT</u> | 1.2 | IUnknown object | Read/write
(Note: Not available to Visual Basic applications) |
| <u>MeetingResponseStatus</u> | 1.2 | Long | Read/write |
| <u>MeetingStatus</u> | 1.2 | Long | Read/write |
| <u>Organizer</u> | 1.2 | AddressEntry object | Read-only |
| <u>Parent</u> | 1.2 | Messages collection object | Read-only |
| <u>ReadReceipt</u> | 1.2 | Boolean | Read/write |
| <u>Recipients</u> | 1.2 | Recipients object or Recipients collection object | Read/write |
| <u>ReminderMinutesBeforeStart</u> | 1.2 | Long | Read/write |
| <u>ReminderSet</u> | 1.2 | Boolean | Read/write |
| <u>ReplyTime</u> | 1.2 | Variant (vbDate format) | Read/write |
| <u>ResponseRequested</u> | 1.2 | Boolean | Read/write |
| <u>Sender</u> | 1.2 | AddressEntry object | Read/write |
| <u>Sensitivity</u> | 1.2 | Long | Read/write |
| <u>Sent</u> | 1.2 | Boolean | Read/write |
| <u>Session</u> | 1.2 | Session object | Read-only |
| <u>Signed</u> | 1.2 | Boolean | Read/write |
| <u>Size</u> | 1.2 | Long | Read-only |
| <u>StartTime</u> | 1.2 | Variant (vbDate format) | Read/write |
| <u>StoreID</u> | 1.2 | String | Read-only |
| <u>Subject</u> | 1.2 | String | Read/write |
| <u>Submitted</u> | 1.2 | Boolean | Read/write |
| <u>Text</u> | 1.2 | String | Read/write |
| <u>TimeCreated</u> | 1.2 | Variant (vbDate format) | Read-only |

| | | | |
|-------------------------|-----|---------------------------------|------------|
| <u>TimeExpired</u> | 1.2 | Variant (vbDate format) | Read/write |
| <u>TimeLastModified</u> | 1.2 | Variant (vbDate format) | Read-only |
| <u>TimeReceived</u> | 1.2 | Variant (vbDate format) | Read/write |
| <u>TimeSent</u> | 1.2 | Variant (vbDate format) | Read/write |
| <u>Type</u> | 1.2 | String | Read/write |
| <u>Unread</u> | 1.2 | Boolean | Read/write |

Methods

| Name | Available since version | Parameters |
|-------------------------------|-------------------------|---|
| <u>ClearRecurrencePattern</u> | 1.2 | (none) |
| <u>CopyTo</u> | 1.2 | <i>folderID</i> as String , (optional) <i>storeID</i> as String |
| <u>Delete</u> | 1.2 | (none) |
| <u>GetRecurrencePattern</u> | 1.2 | (none) |
| <u>IsSameAs</u> | 1.2 | <i>objMessage2</i> as Object |
| <u>MoveTo</u> | 1.2 | <i>folderID</i> as String , (optional) <i>storeID</i> as String |
| <u>Options</u> | 1.2 | (optional) <i>parentWindow</i> as Long |
| <u>Respond</u> | 1.2 | <i>RespondType</i> as Long |
| <u>Send</u> | 1.2 | (optional) <i>saveCopy</i> as Boolean , (optional) <i>showDialog</i> as Boolean , (optional) <i>parentWindow</i> as Long |
| <u>Update</u> | 1.2 | (optional) <i>makePermanent</i> as Boolean , (optional) <i>refreshObject</i> as Boolean |

Remarks

An AppointmentItem object is distinguished from a Message object by its **Type** property containing IPM.Appointment.

New AppointmentItem objects can only be created by using the **Add** method on a Messages collection obtained from a Folder object reserved for calendar data:

```
Dim objSession As Session
Dim objCalendarFolder As Folder
Dim objAppointments As Messages
Dim objNewAppointment As AppointmentItem

Set objCalendarFolder = objSession.GetDefaultFolder _
    (CdoDefaultFolderCalendar)
Set objAppointments = objCalendarFolder.Messages
Set objNewAppointment = objAppointments.Add
```

An appointment can be obtained from its parent Messages collection using the collection's **Item**

property. To get to the Messages collection in a folder, use the Folder object's **Messages** property. If you know an appointment's unique identifier, you can obtain it directly from the Session object's **GetMessage** method.

You can apply a MessageFilter object to a Messages collection containing appointments. However, the current version of CDO only supports filtering on the **EndTime** and **StartTime** properties. An attempt to filter on any other properties, including the inherited Message object properties, returns **CdoE_TOO_COMPLEX**.

You can turn an appointment into a meeting by setting its **MeetingStatus** property to **CdoMeeting** and sending it to one or more recipients. The appointment becomes a meeting as of the moment you call its **Send** method, at which time CDO instantiates a MeetingItem object and identifies you in the appointment's **Organizer** property. The MeetingItem object becomes a member of the Messages collection of each recipient's Inbox. They can treat it programmatically like the Message objects in the collection.

You should not send a meeting request for an appointment to any recipient that represents a distribution list, because the status of its individual members cannot be returned to you. You should only send the meeting request to single messaging users. You can determine if a messaging user is a distribution list by checking the **DisplayType** property of the AddressEntry object representing that user.

To cancel a meeting, you set the **MeetingStatus** property to **CdoMeetingCanceled** and send it to all the recipients. Once you have sent the cancellation you can release the underlying AppointmentItem object, typically by using the **Set** statement to assign it to **Nothing**. Note that simply releasing the appointment does not cancel the meeting, and in fact can produce unexpected behavior. You should always cancel first. Only the organizer of a meeting can cancel or release it.

If you organize and then cancel a meeting when your active calendar store is Microsoft® Outlook™, you must not only release the underlying AppointmentItem object but **Delete** it from the Messages collection as well. Failure to do this could cause unexpected results when working with the folders and their contents.

You can cause an appointment or meeting to become recurring by calling its **GetRecurrencePattern** method, which sets the **IsRecurring** property to **True** and returns a child RecurrencePattern object describing the recurrence characteristics. At first the RecurrencePattern object is populated with the default values indicated in its property descriptions, but you can change them as desired. The appointment or meeting can be restored to nonrecurring status with the **ClearRecurrencePattern** method, which also resets **IsRecurring** to **False**.

Making an appointment recurring does not create any additional AppointmentItem objects for the recurrences. But you may wish to instantiate an individual recurrence so that you can edit it and make it different from other recurrences in the series. You do this by using a MessageFilter object to restrict **CdoPR_START_DATE** and **CdoPR_END_DATE** to the start and end of the desired occurrence, and then calling the Messages collection's **GetFirst** method.

If you instantiate an individual recurrence and change one of its properties, and this property is subsequently modified on the original appointment, the modification to the original is not propagated to your recurrence. However, modifications on the original to properties which you have not changed in your recurrence are automatically propagated.

If any change is made to the original appointment's RecurrencePattern object after you have instantiated one or more individual recurrences, some or all of them may be automatically deleted. Microsoft® Outlook™ deletes all individual recurrences, while Microsoft Schedule+ deletes only those that are earlier than the new **PatternStartDate** or later than the new **PatternEndDate**. CDO does not automatically delete any individual recurrences.

If your calendar folder is in Microsoft Outlook and you change an individual recurrence's **StartTime** or **EndTime** property, the following rules are enforced:

- Two recurrences must not be on the same day.
- A recurrence must not be moved before a previous recurrence or after a subsequent recurrence.
- Recurrences must not be overlapped in time.

If these rules are not followed, Outlook returns **CdoE_COLLISION** on your next call to **Send** or **Update**.

An individual recurrence's **GetRecurrencePattern** method returns the same **RecurrencePattern** object as does the original appointment's **GetRecurrencePattern**. However, the individual recurrence is not the parent of the recurrence pattern. The original AppointmentItem object is the parent and in fact is obtained from the RecurrencePattern object through its **Parent** property:

```
Dim objRecAppt As AppointmentItem ' an individual recurrence
Dim objOrigAppt As AppointmentItem ' the original appointment
Set objOrigAppt = objRecAppt.GetRecurrencePattern.Parent
```

To edit an entire recurring series of appointments, you modify the appropriate properties on either the original AppointmentItem object or its child **RecurrencePattern** object. To edit an individual recurrence only, you instantiate it and modify its AppointmentItem properties. All changes take effect when you call the appointment's **Send** or **Update** method.

The original appointment's **StartTime**, **EndTime**, and **AllDayEvent** properties are disabled whenever its **IsRecurring** property is **True**. Any attempt to access them while in this state returns **CdoE_NO_SUPPORT**. You must use the recurrence pattern's **PatternStartDate**, **PatternEndDate**, **StartTime**, and **EndTime** properties to edit a recurring series.

Calendar folders are not supported in the public folders store provided with Microsoft® Exchange, and AppointmentItem and MeetingItem objects are stored as **Message** objects. An attempt to access a property or method specific to an appointment or meeting, such as **Duration** or **GetAssociatedAppointment**, returns **CdoE_NO_SUPPORT**. If you have declared an AppointmentItem object with early binding, for example Dim objAppt As AppointmentItem, CDO returns **CdoE_NO_SUPPORT** in this case. With late binding, Microsoft® Visual Basic® returns an error indicating no support.

An AppointmentItem object can be rendered into HTML hypertext using the CDO Rendering **ObjectRenderer** object. To specify this, set the object renderer's **DataSource** property to the AppointmentItem object itself. The individual properties that can be rendered with the **RenderProperty** method are indicated in the AppointmentItem and Message object property descriptions.

One or more appointments can also be rendered by the CDO Rendering **CalendarView** object's **RenderAppointments** method. The **ContainerRenderer** object's **DataSource** property must be set to the parent **Messages** collection and its **CurrentView** property to a calendar view.

An AppointmentItem object can also be rendered as the parent of a **Recipients** collection, using the **ContainerRenderer** object. The individual properties that can be rendered with the **RenderProperty** method are indicated in the AppointmentItem and Message object property descriptions.

AllDayEvent Property (AppointmentItem Object) Group

The **AllDayEvent** property indicates whether this appointment is an all-day or multiple-day event. Read/write.

Syntax

objAppointment.AllDayEvent

Data Type

Boolean

Remarks

The **AllDayEvent** property contains **True** if the appointment takes up one or more entire days in the calendar without any free blocks. It contains **False** if the appointment is limited to a specified start time and a duration that is not a multiple of 24 hours.

If you change **AllDayEvent** from **False** to **True**, the appointment's **StartTime** is automatically reset to the midnight (00:00) preceding the current starting date/time, and **EndTime** is reset to the midnight following the current ending date/time. The resulting **Duration** is a nonzero multiple of 24 hours. If the appointment underlies a meeting, these calculations are performed in the current time zone of the appointment's organizer.

If you subsequently change **StartTime** or **EndTime**, you are not required to set them to midnight, but you must set them to the same time of day, such that their values differ by a nonzero multiple of 24 hours. If you do not comply with this restriction while **AllDayEvent** is **True**, you get a return of **CdoE_INVALID_OBJECT** when you call the appointment's **Update** method.

If you change **AllDayEvent** from **True** to **False**, the appointment's **StartTime** is automatically reset to the current value of the "BusinessDayStartTime" option on the current starting date, and **EndTime** is reset to 30 minutes past "BusinessDayStartTime" on the current ending date. Thus the appointment still covers the same number of days as it did before. The "BusinessDayStartTime" option can be obtained from the **Session** object's **GetOption** method and altered with the **SetOption** method. If **AllDayEvent** was already **False**, **StartTime** and **EndTime** are not modified.

If an appointment is viewed through Microsoft Schedule+ and its **StartTime** and **EndTime** are a multiple of 24 hours apart, its **AllDayEvent** property is forced **True** by Schedule+. The starting and ending times are not changed.

If you make this appointment recurring by calling its **GetRecurrencePattern** method, its **StartTime**, **EndTime**, and **AllDayEvent** properties are disabled, and any attempt to access them returns **CdoE_NO_SUPPORT**. Calling **ClearRecurrencePattern** returns the appointment to nonrecurring status, and these properties can once again be accessed. However, their values may have changed if the corresponding properties on the **RecurrencePattern** object were edited while the appointment was recurring.

Calendar folders are not supported in the public folders store provided with Microsoft® Exchange, and AppointmentItem objects are stored as **Message** objects. An attempt to read or write **AllDayEvent** in this case returns **CdoE_NO_SUPPORT**.

Changes you make to properties on an AppointmentItem object take effect when you call the **Send** or **Update** method.

BusyStatus Property (AppointmentItem Object) Group

The **BusyStatus** property returns or sets the busy status of this messaging user for this appointment. Read/write.

Syntax

objAppointment.**BusyStatus**

Data Type

Long

Remarks

The **BusyStatus** property can have exactly one of the following values:

| BusyStatus value | Decimal value | Description |
|-------------------------|----------------------|---|
| CdoFree | 0 | This messaging user has no conflicting commitments during the time span of this appointment. |
| CdoTentative | 1 | This messaging user has at least one tentative commitment during the time span of this appointment. |
| CdoBusy | 2 | This messaging user has at least one confirmed commitment during the time span of this appointment. |
| CdoOutOfOffice | 3 | This messaging user is to be considered out-of-office (OOO) for at least part of the time span of this appointment. |

If two or more conflicting commitments are present during the time span of this appointment, the highest applicable decimal value is returned, representing the most committed state. For example, if the user has one tentative commitment and one confirmed commitment during the time span, the **BusyStatus** property returns **CdoBusy**, that is, the highest level of commitment.

Microsoft Schedule+ only recognizes busy and not busy for the status of a time slot. If an appointment is viewed through Schedule+ and its **BusyStatus** property has a value of **CdoFree** or **CdoTentative**, the status is treated by Schedule+ as not busy. If **BusyStatus** is **CdoBusy** or **CdoOutOfOffice**, Schedule+ treats the status as busy. If an appointment originates in Schedule+, CDO assigns **CdoTentative** to the appropriate time slots.

Calendar folders are not supported in the public folders store provided with Microsoft® Exchange, and AppointmentItem objects are stored as Message objects. An attempt to read or write **BusyStatus** in this case returns **CdoE_NO_SUPPORT**.

Changes you make to properties on an AppointmentItem object take effect when you call the Send or Update method.

ClearRecurrencePattern Method (AppointmentItem Object) Group

The **ClearRecurrencePattern** method removes any recurrence settings from this appointment.

Syntax

objAppointment.ClearRecurrencePattern()

Remarks

The **ClearRecurrencePattern** method sets the **IsRecurring** property to **False** and dissociates this AppointmentItem object from any RecurrencePattern object it might have had assigned to it.

ClearRecurrencePattern calls **Release** on the RecurrencePattern object. This is normally the final **Release** because the RecurrencePattern object applies only to its parent appointment and cannot be used for any other AppointmentItem object. The RecurrencePattern object is removed from memory in response to its final **Release**.

The **ClearRecurrencePattern** method is only valid on a nonrecurring appointment or an appointment originating a recurring series. An attempt to call it on an individual recurrence in a series returns **CdoE_NO_SUPPORT**.

Calendar folders are not supported in the public folders store provided with Microsoft® Exchange, and AppointmentItem objects are stored as Message objects. An attempt to call **ClearRecurrencePattern** in this case returns **CdoE_NO_SUPPORT**.

The effect of the **ClearRecurrencePattern** operation is not permanent until you call the **Update** method on the AppointmentItem object.

Calling **ClearRecurrencePattern** does not immediately invalidate references to individual recurrences that have been instantiated. You can still access the properties and methods of an individual recurrence, but when you call **Update** to make your changes persistent, you get a return of **CdoE_INVALID_OBJECT**.

Duration Property (AppointmentItem Object) Group

The **Duration** property returns the duration of this appointment in minutes. Read-only.

Syntax

objAppointment.Duration

Data Type

Long

Remarks

The **Duration** property contains the number of minutes the appointment is to last. The minimum value is 0 and the maximum value is 1,490,000, which is treated as infinity.

Since **Duration** is read-only, you must change the **StartTime** or **EndTime** property to cause a new value to be calculated for **Duration**.

Among its possible values, the **Duration** property can be any multiple of 24 hours that does not exceed its maximum value. If an appointment has such a value for **Duration**, Microsoft® Schedule+ interprets it as lasting exactly 24 hours. That is, Schedule+ treats **Duration** values of 0, 1440, 2880, 4320, and so on as if they were 1440.

Calendar folders are not supported in the public folders store provided with Microsoft® Exchange, and AppointmentItem objects are stored as Message objects. An attempt to read **Duration** in this case returns **CdoE_NO_SUPPORT**.

Example

This code fragment causes CDO to calculate a value of 90 minutes for the **Duration** property of an appointment:

```
Dim objAppt As Appointment
' ...
objAppt.EndTime = DateAdd("n", 90, objAppt.StartTime) ' n for minutes!
```


EndTime Property (AppointmentItem Object) Group

The **EndTime** property returns or sets the ending date/time of this appointment. Read/write.

Syntax

objAppointment.**EndTime**

Data Type

Variant (**vbDate** format)

Remarks

The **EndTime** property uses both the date and the time component of the **vbDate** format. Its default value is the current time rounded up to the next half hour.

The **EndTime** property ignores seconds and truncates the time component to the minute.

The value of **EndTime** must be greater than that of the **StartTime** property. If you make settings that violate this constraint, your next call to the **Update** method returns **CdoE_INVALID_OBJECT**.

If you change the value of **EndTime**, **Duration** is automatically recalculated from the new **EndTime** and unchanged **StartTime** values.

You can apply a **MessageFilter** object to a **Messages** collection containing **AppointmentItem** objects and filter them on the **StartTime** and **EndTime** properties. The filter passes appointments starting on or before the date/time in **CdoPR_START_DATE** and ending on or after the date/time in **CdoPR_END_DATE**.

If you make this appointment recurring by calling its **GetRecurrencePattern** method, its **StartTime**, **EndTime**, and **AllDayEvent** properties are disabled, and any attempt to access them returns **CdoE_NO_SUPPORT**. Calling **ClearRecurrencePattern** returns the appointment to nonrecurring status, and these properties can once again be accessed. However, their values may have changed if the corresponding properties on the **RecurrencePattern** object were edited while the appointment was recurring.

The appointment's **StartTime** and **EndTime** properties are always held internally in UTC (Coordinated Universal Time, also known as GMT). By contrast, the **RecurrencePattern** object's **StartTime** and **EndTime** properties are always held internally in the organizer's current time zone. However, all these properties are converted to the local messaging user's current time zone whenever they are displayed or read programmatically.

Calendar folders are not supported in the public folders store provided with Microsoft® Exchange, and **AppointmentItem** objects are stored as **Message** objects. An attempt to read or write **EndTime** in this case returns **CdoE_NO_SUPPORT**.

Changes you make to properties on an **AppointmentItem** object take effect when you call the **Send** or **Update** method.

The **EndTime** property corresponds to the MAPI property PR_END_DATE.

GetRecurrencePattern Method (AppointmentItem Object)

Group

The **GetRecurrencePattern** method returns a [RecurrencePattern](#) object defining the recurrence attributes of an appointment.

Syntax

Set *objRecurPatt* = *objAppointment*.**GetRecurrencePattern**()

objRecurPatt

On successful return, contains the RecurrencePattern object.

objAppointment

Required. This AppointmentItem object.

Remarks

If the appointment had already been specified as recurring, the **GetRecurrencePattern** method returns the [RecurrencePattern](#) object containing the current recurrence characteristics. If the appointment was previously nonrecurring, it is made recurring and **GetRecurrencePattern** returns a new RecurrencePattern object populated with the default values indicated in its property descriptions.

The [IsRecurring](#) property is set to **True** when the **GetRecurrencePattern** method is called.

Note You cannot use **GetRecurrencePattern** to test whether an appointment is recurring, because it always returns a RecurrencePattern object and forces the appointment to be recurring. Instead, you should test for recurrence using the **IsRecurring** property.

You can use the [ClearRecurrencePattern](#) method to return the appointment to nonrecurring status.

Calendar folders are not supported in the public folders store provided with Microsoft® Exchange, and AppointmentItem objects are stored as [Message](#) objects. An attempt to call **ClearRecurrencePattern** in this case returns **CdoE_NO_SUPPORT**.

Example

This code fragment makes an appointment recurring and changes its [Subject](#) property. It is equally valid for a nonrecurring appointment, an appointment originating a recurring series, and an individual recurrence in that series. If *objAppt* is nonrecurring before executing this fragment, it becomes the originator of a recurring series. If it is already an originator, doing *objOrig.Update* is equivalent to doing *objAppt.Update*.

```
Dim objAppt, objOrig As AppointmentItem
Dim objRecPatt As RecurrencePattern
' objAppt can be solitary, an originator, or a recurrence
Set objRecPatt = objAppt.GetRecurrencePattern
With objRecPatt
    ' set recurrence pattern properties as desired
End with
Set objOrig = objRecPatt.Parent
With objOrig
    .Subject = "New subject for entire recurring series"
    .Update ' necessary for any changes to take effect
End with
```

Note If *objAppt* in the preceding fragment is an individual recurrence, its properties are not

changed by *objOrig*.**Update**. This is because changes you make in the original appointment only affect recurrences instantiated after the update. Because *objAppt* was already instantiated before the update, its **Subject** property is not updated when *objOrig*.**Subject** is. To avoid this problem, you can refresh *objAppt* by repeating the **Set** statement that instantiated it in the first place.

IsRecurring Property (AppointmentItem Object) Group

The **IsRecurring** property indicates whether this appointment is specified as recurring. Read-only.

Syntax

objAppointment.IsRecurring

Data Type

Boolean

Remarks

The **IsRecurring** property contains **True** if the appointment is recurring and **False** if it is not. **IsRecurring** defaults to **False** in a newly created AppointmentItem object.

IsRecurring is set to **True** when the [GetRecurrencePattern](#) method is called and to **False** when the [ClearRecurrencePattern](#) method is called.

Calendar folders are not supported in the public folders store provided with Microsoft® Exchange, and AppointmentItem objects are stored as [Message](#) objects. An attempt to read **IsRecurring** in this case returns **CdoE_NO_SUPPORT**.

Location Property (AppointmentItem Object) Group

The **Location** property returns or sets the location of this appointment. Read/write.

Syntax

objAppointment.**Location**

Data Type

String

Remarks

The **Location** property contains a string representing the specific location in which this appointment is scheduled, such as an office or conference room. It is typically set by the messaging user that creates the MeetingItem object from the appointment. The initiating messaging user is available through the Organizer property.

Calendar folders are not supported in the public folders store provided with Microsoft® Exchange, and AppointmentItem objects are stored as Message objects. An attempt to read or write **Location** in this case returns **CdoE_NO_SUPPORT**.

Changes you make to properties on an AppointmentItem object take effect when you call the Send or Update method.

The **Location** property corresponds to the MAPI property PR_OFFICE_LOCATION.

MeetingResponseStatus Property (AppointmentItem Object) Group

The **MeetingResponseStatus** property returns or sets the overall status of this appointment for this messaging user. Read/write.

Syntax

objAppointment.**MeetingResponseStatus**

Data Type

Long

Remarks

You turn an appointment into a meeting by setting its **MeetingStatus** property to **CdoMeeting** and sending it to one or more recipients. Their responses to your meeting request come back to you in the form of **MeetingItem** objects with the **MeetingType** property set to **CdoMeetingResponse**. You can call **GetAssociatedAppointment** on each meeting response and read the returned appointment's **MeetingResponseStatus** property to find out the response of that individual recipient.

The **MeetingResponseStatus** property can have exactly one of the following values:

| MeetingResponseStatus value | Decimal value | Description |
|------------------------------------|----------------------|---|
| CdoResponseNone | 0 | Used by Microsoft® Outlook™ to indicate that this messaging user has not responded to the meeting request. |
| CdoResponseAccepted | 3 | This messaging user has responded to the meeting request with a firm acceptance. |
| CdoResponseDeclined | 4 | This messaging user has responded to the meeting request with a declination. |
| CdoResponseTentative | 2 | This messaging user has responded to the meeting request with a tentative acceptance. |
| CdoResponseOrganized | 1 | Used by Microsoft® Schedule+ to indicate that this messaging user has not responded to the meeting request. |

You can get the meeting status for each **Recipient** object from its **MeetingResponseStatus** property.

Calendar folders are not supported in the public folders store provided with Microsoft® Exchange, and AppointmentItem objects are stored as **Message** objects. An attempt to read or write **MeetingResponseStatus** in this case returns **CdoE_NO_SUPPORT**.

Changes you make to properties on an AppointmentItem object take effect when you call the **Send** or **Update** method.

MeetingStatus Property (AppointmentItem Object)

Group

The **MeetingStatus** property returns or sets the overall meeting status of this appointment. Read/write.

Syntax

objAppointment.MeetingStatus

Data Type

Long

Remarks

The **MeetingStatus** property is used to indicate whether this appointment represents a meeting, and if so, what its status is.

MeetingStatus can have exactly one of the following values:

| MeetingStatus value | Decimal value | Description |
|---------------------------|---------------|--|
| CdoNonMeeting | 0 | This appointment has been scheduled by this messaging user alone and does not represent a meeting (default value). |
| CdoMeeting | 1 | The appointment has been or is to be made into a meeting. |
| CdoMeetingCanceled | 5 | The meeting organizer has canceled the meeting. |
| CdoMeetingReceived | 3 | The requests for the meeting have been received by the intended attendees. |

When an AppointmentItem object is first created, its **MeetingStatus** property contains **CdoNonMeeting**. The appointment becomes a meeting when you set **MeetingStatus** to **CdoMeeting** and **Send** it to one or more recipients. This creates a MeetingItem object for the meeting and identifies you as the **Organizer**. The meeting request is usually in your Sent Items folder, and the meeting responses from the recipients appear in your Inbox.

You should not send a meeting request to any recipient that represents a distribution list, because the status of its individual members cannot be returned to you. You should only send the meeting request to single messaging users. You can determine if a messaging user is a distribution list by checking the **DisplayType** property of the AddressEntry object representing that user.

To cancel a meeting, you set the **MeetingStatus** property to **CdoMeetingCanceled** and send it to all the recipients. Once you have sent the cancellation you can release the underlying AppointmentItem object, typically by using the **Set** statement to assign it to **Nothing**. Note that simply releasing the appointment does not cancel the meeting, and in fact can produce unexpected behavior. Only the organizer of a meeting can cancel or release it.

If you organize and then cancel a meeting when your active calendar store is Microsoft® Outlook™, you

must not only release the underlying AppointmentItem object but **Delete** it from the Messages collection as well. Failure to do this could cause unexpected results when working with the folders and their contents.

Before you call the appointment's **Send** method, you must set **MeetingStatus** to **CdoMeeting** or **CdoMeetingCanceled**. If **MeetingStatus** contains any other value or is not set, **Send** returns **CdoE_NO_SUPPORT**.

Calendar folders are not supported in the public folders store provided with Microsoft® Exchange, and AppointmentItem objects are stored as Message objects. An attempt to read or write **MeetingStatus** in this case returns **CdoE_NO_SUPPORT**.

Changes you make to properties on an AppointmentItem object take effect when you call the **Send** or **Update** method.

Organizer Property (AppointmentItem Object) Group

The **Organizer** property returns the messaging user that initiated a meeting from this appointment. Read-only.

Syntax

objAppointment.**Organizer**

Data Type

Object (AddressEntry)

Remarks

The **Organizer** property returns an [AddressEntry](#) object representing the messaging user that created the [MeetingItem](#) object from this appointment.

The **Organizer** property is automatically set at the moment the AppointmentItem object is created.

Calendar folders are not supported in the public folders store provided with Microsoft® Exchange, and AppointmentItem objects are stored as [Message](#) objects. An attempt to read **Organizer** in this case returns **CdoE_NO_SUPPORT**.

ReminderMinutesBeforeStart Property (AppointmentItem Object) Group

The **ReminderMinutesBeforeStart** property indicates how many minutes before the start of this appointment a reminder should be issued. Read/write.

Syntax

objAppointment.ReminderMinutesBeforeStart

Data Type

Long

Remarks

The **ReminderMinutesBeforeStart** property must contain a positive integer. It is not enabled unless the **ReminderSet** property contains **True**. **ReminderMinutesBeforeStart** defaults to 15 minutes in a newly created AppointmentItem object.

You should not attempt to access **ReminderMinutesBeforeStart** without first verifying that **ReminderSet** is **True**. If **ReminderSet** is **False**, reading or writing **ReminderMinutesBeforeStart** returns **CdoE_NOT_FOUND**.

The **ReminderMinutesBeforeStart** property can theoretically be set as high as 2,147,483,647, which represents more than 4000 years. However, CDO checks that the designated reminder date/time is valid, that is, not earlier than January 1, 1601, the earliest date/time that can be represented in Microsoft® Windows NT®. If **ReminderMinutesBeforeStart** is outside the limits of validity, CDO returns **CdoE_INVALID_OBJECT** when you call the **Send** or **Update** method.

Note If you set the **ReminderMinutesBeforeStart** property on an appointment and then make the appointment into a meeting by sending it to one or more recipients, the value of **ReminderMinutesBeforeStart** for each recipient depends on how that recipient processes the meeting request. If the recipient uses a CDO Library application or Microsoft® Outlook™, your setting of **ReminderMinutesBeforeStart** is preserved. If the recipient uses Microsoft® Schedule+, **ReminderMinutesBeforeStart** is reset to the default value of 15 minutes. In either case, the value can be altered as desired on the recipient's version of the AppointmentItem object.

Schedule+ also resets **ReminderMinutesBeforeStart** to 15 minutes for any recipient that resets **ReminderSet** to **False**.

Calendar folders are not supported in the public folders store provided with Microsoft® Exchange, and AppointmentItem objects are stored as Message objects. An attempt to read or write **ReminderMinutesBeforeStart** in this case returns **CdoE_NO_SUPPORT**.

Changes you make to properties on an AppointmentItem object take effect when you call the **Send** or **Update** method.

ReminderSet Property (AppointmentItem Object)

Group

The **ReminderSet** property indicates whether this messaging user is to be reminded of this appointment. Read/write.

Syntax

objAppointment.ReminderSet

Data Type

Boolean

Remarks

The **ReminderSet** property contains **True** if a reminder has been set for this appointment and **False** otherwise. **ReminderSet** defaults to **True** in a newly created AppointmentItem object.

The **ReminderMinutesBeforeStart** property is not enabled unless the **ReminderSet** property contains **True**.

Calendar folders are not supported in the public folders store provided with Microsoft® Exchange, and AppointmentItem objects are stored as **Message** objects. An attempt to read or write **ReminderSet** in this case returns **CdoE_NO_SUPPORT**.

Changes you make to properties on an AppointmentItem object take effect when you call the **Send** or **Update** method.

ReplyTime Property (AppointmentItem Object) Group

The **ReplyTime** property returns or sets the date/time a recipient replies to the meeting request associated with this appointment. Read/write.

Syntax

objAppointment.ReplyTime

Data Type

Variant (**vbDate** format)

Remarks

The **ReplyTime** property is not meaningful on the original AppointmentItem object held in the calendar folder of the meeting's **Organizer**. Instead, **ReplyTime** is used on the AppointmentItem object created in the calendar folder of each meeting request recipient that calls **Respond** with a parameter of either **CdoResponseAccepted** or **CdoResponseTentative**. The recipient can access the associated appointment with the **MeetingItem** object's **GetAssociatedAppointment** method.

The **ReplyTime** property does not have a default value. If you attempt to read it when it has never been set, it returns **CdoE_NOT_FOUND**. The current version of CDO does not set **ReplyTime**, so your application should set it when you reply to the meeting request:

```
Dim objMtgReq As MeetingItem ' received meeting request
Dim objMtgRsp As MeetingItem ' response to meeting request
Dim objAssocAppt As AppointmentItem ' associated appointment
Set objMtgRsp = objMtgReq.Respond(CdoResponseAccepted)
Set objAssocAppt = objMtgRsp.GetAssociatedAppointment
objAssocAppt.ReplyTime = Now ' log response time
objMtgRsp.Send ' saves a copy of the response by default
```

Calendar folders are not supported in the public folders store provided with Microsoft® Exchange, and AppointmentItem objects are stored as **Message** objects. An attempt to read or write **ReplyTime** in this case returns **CdoE_NO_SUPPORT**.

Changes you make to properties on an AppointmentItem object take effect when you call the **Send** or **Update** method.

Respond Method (AppointmentItem Object) Group

The **Respond** method returns a MeetingItem object for responding to a meeting request for an appointment.

Syntax

Set *objMeetResp* = *objAppointment*.**Respond**(*RespondType*)

objMeetResp

Object. On successful return, contains a MeetingItem object that can be used to respond to the meeting request.

objAppointment

Required. An AppointmentItem object returned by the GetAssociatedAppointment method of a MeetingItem object.

RespondType

Required. Long. The value to send as the response.

Remarks

The **Respond** method prepares a meeting response which can be sent in answer to a meeting request using the Send method. The MeetingItem object has as a primary recipient the messaging user that created the requesting MeetingItem object from this appointment. The initiating user is available through the Organizer property.

The *RespondType* parameter can have exactly one of the following values:

| <i>RespondType</i> setting | Decimal value | Description |
|-----------------------------|---------------|---|
| CdoResponseAccepted | 3 | This messaging user wishes to firmly accept the meeting request. |
| CdoResponseDeclined | 4 | This messaging user wishes to decline the meeting request. |
| CdoResponseTentative | 2 | This messaging user wishes to tentatively accept the meeting request. |

The message class of the response you send depends on the value you specify in the *RespondType* parameter. It is IPM.Schedule.Meeting.Resp.Pos if you accept, IPM.Schedule.Meeting.Resp.Neg if you decline, or IPM.Schedule.Meeting.Resp.Tent if you accept tentatively.

The only AppointmentItem objects on which you can call the **Respond** method are those returned by the GetAssociatedAppointment method of a MeetingItem object. An attempt to call **Respond** on any other AppointmentItem object, such as one obtained directly from a calendar folder, returns **CdoE_NO_SUPPORT**.

You can respond to the same meeting request more than once. If you call the **Respond** method with the the *RespondType* parameter set to **CdoResponseAccepted** or **CdoResponseTentative**, CDO creates an AppointmentItem object associated with the meeting and saves it in your active calendar

folder. You can use the **Respond** method on this appointment to change your response.

If you **Respond** to a meeting request with **CdoResponseDeclined**, no AppointmentItem object is created, but any AppointmentItem already in the folder is left undeleted. Therefore, if you accept a request and subsequently decline it, you must either **Delete** the associated AppointmentItem object yourself or leave it in the folder.

If you have declined a meeting request and subsequently wish to accept it, you cannot use any associated AppointmentItem object for your new response. However, if you have retained the requesting MeetingItem object, you can use its **Respond** method to accept the request.

If a meeting request is for a recurring meeting, your response applies to the entire recurring series. If you respond with **CdoResponseAccepted** or **CdoResponseTentative**, you can subsequently select an individual recurrence and alter your response to **CdoResponseAccepted**, **CdoResponseTentative**, or **CdoResponseDeclined**.

You cannot call **Respond** on a meeting cancellation, that is, a meeting request with **CdoMeetingCanceled** in its associated appointment's MeetingStatus property.

Calendar folders are not supported in the public folders store provided with Microsoft® Exchange, and AppointmentItem objects are stored as Message objects. An attempt to call **Respond** in this case returns **CdoE_NO_SUPPORT**.

Calling **GetAssociatedAppointment** on the MeetingItem object and then calling the appointment's **Respond** method is the same as calling the **Respond** method directly on the MeetingItem object.

Example

See the example for the MeetingItem object's **Respond** method.

ResponseRequested Property (AppointmentItem Object)

Group

The **ResponseRequested** property indicates whether a response is requested for this appointment. Read/write.

Syntax

objAppointment.ResponseRequested

Data Type

Boolean

Remarks

The **ResponseRequested** property contains **True** if the messaging user that created a [MeetingItem](#) object from this appointment wishes a response. It contains **False** if there is no request for a response. **ResponseRequested** defaults to **True**.

Calendar folders are not supported in the public folders store provided with Microsoft® Exchange, and AppointmentItem objects are stored as [Message](#) objects. An attempt to read or write **ResponseRequested** in this case returns **CdoE_NO_SUPPORT**.

Changes you make to properties on an AppointmentItem object take effect when you call the [Send](#) or [Update](#) method.

StartTime Property (AppointmentItem Object) Group

The **StartTime** property returns or sets the starting date/time of this appointment. Read/write.

Syntax

objAppointment.**StartTime**

Data Type

Variant (**vbDate** format)

Remarks

The **StartTime** property uses both the date and the time component of the **vbDate** format. Its default value is the current time rounded down to the preceding half hour.

The **StartTime** property ignores seconds and truncates the time component to the minute.

The value of **StartTime** must be less than that of the **EndTime** property. If you make settings that violate this constraint, your next call to the **Update** method returns **CdoE_INVALID_OBJECT**.

If you change the value of **StartTime**, **Duration** is automatically recalculated from the new **StartTime** and unchanged **EndTime** values.

You can apply a **MessageFilter** object to a **Messages** collection containing **AppointmentItem** objects and filter them on the **StartTime** and **EndTime** properties. The filter passes appointments starting on or before the date/time in **CdoPR_START_DATE** and ending on or after the date/time in **CdoPR_END_DATE**.

If you make this appointment recurring by calling its **GetRecurrencePattern** method, its **StartTime**, **EndTime**, and **AllDayEvent** properties are disabled, and any attempt to access them returns **CdoE_NO_SUPPORT**. Calling **ClearRecurrencePattern** returns the appointment to nonrecurring status, and these properties can once again be accessed. However, their values may have changed if the corresponding properties on the **RecurrencePattern** object were edited while the appointment was recurring.

The appointment's **StartTime** and **EndTime** properties are always held internally in UTC (Coordinated Universal Time, also known as GMT). By contrast, the **RecurrencePattern** object's **StartTime** and **EndTime** properties are always held internally in the organizer's current time zone. However, all these properties are converted to the local messaging user's current time zone whenever they are displayed or read programmatically.

Calendar folders are not supported in the public folders store provided with Microsoft® Exchange, and **AppointmentItem** objects are stored as **Message** objects. An attempt to read or write **StartTime** in this case returns **CdoE_NO_SUPPORT**.

Changes you make to properties on an **AppointmentItem** object take effect when you call the **Send** or **Update** method.

The **StartTime** property corresponds to the MAPI property **PR_START_DATE**.

Attachment Object

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

At a Glance

| | |
|----------------------------|-------------------------------|
| Specified in type library: | CDO.DLL |
| First available in: | CDO Library version 1.0.a |
| Parent objects: | <u>Attachments</u> collection |
| Child objects: | <u>Fields</u> collection |
| Default property: | <u>Name</u> |

Properties

| Name | Available since version | Type | Access |
|--------------------|-------------------------|--|--|
| <u>Application</u> | 1.0.a | String | Read-only |
| <u>Class</u> | 1.0.a | Long | Read-only |
| <u>Fields</u> | 1.1 | Field object or Fields collection object | Read-only |
| <u>Index</u> | 1.0.a | Long | Read-only |
| <u>MAPIOBJECT</u> | 1.1 | IUnknown object | Read/write
(Note: Not available to Visual Basic applications) |
| <u>Name</u> | 1.0.a | String | Read/write |
| <u>Parent</u> | 1.0.a | Attachments collection object | Read-only |
| <u>Position</u> | 1.0.a | Long | Read/write |
| <u>Session</u> | 1.0.a | Session object | Read-only |
| <u>Source</u> | 1.0.a | String or Message object | Read/write |
| <u>Type</u> | 1.0.a | Long | Read/write |

Methods

| Name | Available since version | Parameters |
|---------------------|-------------------------|------------------------------------|
| <u>Delete</u> | 1.0.a | (none) |
| <u>IsSameAs</u> | 1.1 | <i>objAttach2</i> as Object |
| <u>ReadFromFile</u> | 1.0.a | <i>fileName</i> as String |
| <u>WriteToFile</u> | 1.0.a | <i>fileName</i> as String |

Remarks

An attachment is an object, such as a file or an OLE object, that is associated with and transmitted with a Message object. It is assigned a particular location within the message, specified by the **Position** property, and overwrites the character at that position when the message is displayed to a messaging user. Typically, a placeholder such as an icon is displayed instead of the attachment's contents, until the user requests that the attachment be opened and displayed in its entirety.

The Microsoft® CDO Library does not manage the actual display of the attachment or its placeholder. The properties of the Attachment object simply provide information which the displaying application can use to find and open the attachment, select a suitable placeholder, and convert the attachment's contents into a display.

An Attachment object can be rendered into HTML hypertext using the CDO Rendering ObjectRenderer object. To specify this, set the object renderer's **DataSource** property to the Attachment object itself. The individual properties that can be rendered with the **RenderProperty** method are indicated in the Attachment object property descriptions.

Delete Method (Attachment Object)

Group

The **Delete** method removes the Attachment object from the [Attachments](#) collection.

Syntax

objAttachment.Delete()

Remarks

The **Delete** method performs an irreversible operation on the collection. It calls **Release** on the collection's reference to the Attachment object. If you have another reference to the attachment, you can still access its properties and methods, but you can never again associate it with any collection because the **Add** method always creates a new object. You should **Set** your reference variable either to **Nothing** or to another attachment.

The final **Release** on the Attachment object takes place when you assign your reference variable to **Nothing**, or when you call **Delete** if you had no other reference. At this point the object is removed from memory. Attempted access to a released object results in an error return of **CdoE_INVALID_OBJECT**.

When you delete a member of a collection, the collection is immediately refreshed, meaning that its **Count** property is reduced by one and its members are reindexed. To access a member following the deleted member, you must use its new index value. For more information, see [Looping Through a Collection](#).

The effect of the **Delete** operation is not permanent until you use the **Update**, **Send**, or **Delete** method on the [Message](#) object to which this attachment belongs.

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.

Example

This code fragment illustrates the two situations previously explained. The **Set** statement calls **AddRef** on the first Attachment object. That reference survives the call to **Delete** and has to be reassigned. The second Attachment object is deleted without creating another reference, and no other action is necessary.

```
' assume valid Message object
Set objAttachment = objMessage.Attachments.Item(1)
objAttachment.Delete ' still have a reference from Set statement
' ... other operations on objAttachment possible but pointless ...
Set objAttachment = Nothing ' necessary to remove reference
' ...
objMessage.Attachments.Item(2).Delete ' no reference to remove
```

Fields Property (Attachment Object)

Group

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

Integer. Value must be greater than 0 and less than or equal to 65,535 (&HFFFF). Specifies the index within the Fields collection.

proptag

Long. Value must be greater than or equal to 65,536 (&H10000). 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 (Field or Fields collection)

Remarks

The **Fields** property returns one or all of the fields associated with an Attachment object. Each field typically corresponds to a MAPI property. Data types are preserved, except that MAPI counted binary properties are converted to and from character strings representing hexadecimal digits.

The **Fields** property provides a generic access mechanism that allows Microsoft® Visual Basic® and Microsoft® 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 **CdoPR_ATTACH_SIZE**. To access a named property, use *objAttachment.Fields(name)*, where *name* is a string that represents the custom property name.

Although the **Fields** property itself is read-only, the collection it returns can be accessed in the normal manner through its **Add** and **Delete** methods, and the properties on its member [Field](#) objects retain their respective read/write or read-only accessibility.

The **Fields** property does not correspond to a MAPI property and cannot be rendered into HTML hypertext by the CDO Rendering Library.

Index Property (Attachment Object)

Group

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 be saved and used later with the collection's **Item** property to reselect the same attachment in the collection.

The first object in the collection has an **Index** value of 1.

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.

The **Index** property does not correspond to a MAPI property and cannot be rendered into HTML hypertext by the CDO Rendering Library.

Example

```
Function Attachments_GetByIndex()  
Dim lIndex As Long  
Dim objOneAttach As Attachment ' 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 it 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)

Group

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

Syntax

boolSame = *objAttachment*.**IsSameAs**(*objAttach2*)

boolSame

On successful return, contains **True** if the two objects are the same.

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 in the underlying messaging system. 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. This is necessary because, although MAPI requires all entry identifiers to be unique, it does not require two of them identifying the same object to be identical. A generic comparison of any two objects' unique identifiers is also available with the Session object's **CompareIDs** method.

MAPIOBJECT Property (Attachment Object) Group

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

Syntax

objAttachment.MAPIOBJECT

Data Type

Variant (**vbDataObject** format)

Remarks

The **MAPIOBJECT** property is not available to Visual Basic programs. It is accessible only by C/C++ programs that deal with **IUnknown** objects. Visual Basic supports the **IDispatch** interface and not **IUnknown**. The **MAPIOBJECT** property is an **IUnknown** object that returns an **IAttach** interface in response to **QueryInterface**. For more information, see [Introduction to Automation](#) and [How Programmable Objects Work](#). Also see the "COM and ActiveX Object Services" section of the Microsoft Platform SDK.

If your application uses any **MAPIOBJECT** or **RawTable** properties, it must **Release** them all before calling the [Session](#) object's **Logoff** method. Failure to do so can result in unexpected behavior.

The **MAPIOBJECT** property does not correspond to a MAPI property and cannot be rendered into HTML hypertext by the CDO Rendering Library.

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**

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

Data Type

String

Remarks

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

If the attachment's name is not supplied at creation time and is not set subsequently, **Name** returns the following values depending on the setting of the attachment's **Type** property:

| Attachment Type property | Attachment Name property if never set |
|---------------------------------|--|
| CdoFileData | The file name from the attachment's Source property |
| CdoFileLink | The file name from the attachment's Source property |
| CdoOLE | An empty string |
| CdoEmbeddedMessage | An empty string |

The **Name** property corresponds to the MAPI property PR_DISPLAY_NAME. It can be rendered into HTML hypertext using the CDO Rendering ObjectRenderer object. To specify this, set the object renderer's **DataSource** property to this Attachment object and the *property* parameter of the **RenderProperty** method to **CdoPR_DISPLAY_NAME**.

Example

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

Position Property (Attachment Object)

Group

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 displayed 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.

A positive value of **Position** represents an index to the character within the message text to be replaced by the attachment. The first text character has an index of 1. The value 0 indicates that the attachment is present but should not be made visible in the displayed message. The value -1 indicates that the attachment is not handled using the **Position** property.

CDO does not manage the actual display of the attachment within the message. The **Position** property simply provides a location for the displaying application, which must find and replace the appropriate character in the message's **Text** property.

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

The **Position** property corresponds to the MAPI property PR_RENDERING_POSITION. It can be rendered into HTML hypertext using the CDO Rendering ObjectRenderer object. To specify this, set the object renderer's **DataSource** property to this Attachment object and the *property* parameter of the RenderProperty method to **CdoPR_RENDERING_POSITION**.

Note The MAPI rendering position is zero-based, meaning that the first character in the message text is considered to be at position zero. All CDO string indexes, including the **Position** property, are one-based. If you render the **Position** property into HTML, the MAPI value is used for the rendering instead of the CDO value. This could potentially lead to confusion regarding which message text character is to be replaced with the attachment.

Example

```
' from the function Attachments_Add()  
  Set objAttach = objAttachColl.Add ' add an attachment  
  With objAttach  
    .Type = CdoFileLink  
    .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 CdoFileLink"
```

ReadFromFile Method (Attachment Object) Group

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

Syntax

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

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 |
|---------------------------|---|
| CdoFileData | Copies the contents of the specified file to the attachment. |
| CdoFileLink | (Not supported) |
| CdoOLE | Reads the attachment from the specified file, which must be in OLE docfile format. The file could have been previously written by the WriteToFile method with an CdoOLE type setting. |
| CdoEmbeddedMessage | (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 The current version of CDO does not support **ReadFromFile** for **CdoFileLink** or **CdoEmbeddedMessage** attachments. These calls generate the run-time error **CdoE_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 information specifying the location of the data for the attachment. Read/write.

Syntax

objAttachment.**Source**

Data Type

String or Object (Message)

Remarks

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

Note that the **Source** property is a string except when it returns the source of an **CdoEmbeddedMessage** attachment.

CDO does not synchronize the **Source** property and the ReadFromFile method. For **CdoFileData** and **CdoOLE** 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 or setting of the **Source** property depends on the value of the **Type** property, as described in the following table:

| Type property | Source property |
|---------------------------|--|
| CdoFileData | Specifies a full path and file name that contains the data for the attachment, for example C:\DOCUMENT\BUDGET.XLS. |
| CdoFileLink | Specifies a full path and file name in a universal naming convention (UNC) format, such as \\SALES\INFO\PRODUCTS\NEWS.DOC. |
| CdoOLE | Specifies a full path and file name to a valid OLE docfile, for example C:\DOCUMENT\BUDGET2.XLS. |
| CdoEmbeddedMessage | Specifies the unique identifier of the message to be embedded; returns the embedded <u>Message</u> object. |

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

Note You must set **Type** before you set **Source**. Failure to do this can result in a return of **CdoE_NOT_FOUND** from the ReadFromFile or WriteToFile method.

The **Source** property can also be set at the time of creation of the attachment by supplying the *source* parameter to the **Add** method of the Attachments collection. For attachments of type **CdoFileData**, the

Add method is the only place the source file can be specified. However, you can change it later with the Attachment object's **ReadFromFile** method.

You should always set **Source** before displaying the attachment, because some calendaring clients, such as Microsoft® Outlook™, return an error message when you attempt to display an AppointmentItem, MeetingItem, or Message object with an empty attachment, that is, an attachment with no properties set.

The **Source** property corresponds to the MAPI property PR_ATTACH_PATHNAME. It can be rendered into HTML hypertext using the CDO Rendering ObjectRenderer object. To specify this, set the object renderer's **DataSource** property to this Attachment object and the *property* parameter of the **RenderProperty** method to **CdoPR_ATTACH_PATHNAME**.

Example

```
' from the function Attachments_Add()
  Set objAttach = objAttachColl.Add ' add an attachment
  With objAttach
    .Type = CdoFileLink
    .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 CdoFileLink"
```

Type Property (Attachment Object)

Group

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 |
|---------------------------|-------|---|
| CdoFileData | 1 | Attachment is the contents of a file.
(Default value.) |
| CdoFileLink | 2 | Attachment is a link to a file. |
| CdoOLE | 3 | Attachment is an OLE object. |
| CdoEmbeddedMessage | 4 | Attachment is an embedded message. |

The value of the **Type** property determines the valid values for the **Source** property. Consequently, you must set **Type** before setting **Source** in order for the [ReadFromFile](#) and [WriteToFile](#) methods to work correctly.

The **Type** property can also 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. It can be rendered into HTML hypertext using the CDO Rendering [ObjectRenderer](#) object. To specify this, set the object renderer's [DataSource](#) property to this Attachment object and the *property* parameter of the [RenderProperty](#) method to **CdoPR_ATTACH_METHOD**.

Example

```
' from the function Attachments_Add()  
  Set objAttach = objAttachColl.Add ' add an attachment  
  With objAttach  
    .Type = CdoFileLink  
    .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 CdoFileLink"
```

WriteToFile Method (Attachment Object) Group

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

Syntax

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

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 |
|---------------------------|--|
| CdoFileData | Copies the contents of the attachment to the specified file. |
| CdoFileLink | (Not supported) |
| CdoOLE | Writes the attachment to the specified file in OLE docfile format. The file can subsequently be read by the ReadFromFile method with an CdoOLE type setting. |
| CdoEmbeddedMessage | (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 The current version of CDO does not support **WriteToFile** for **CdoFileLink** or **CdoEmbeddedMessage** attachments. These calls generate the run-time error **CdoE_NO_SUPPORT**.

Attachments Collection Object

The Attachments collection object contains one or more Attachment objects.

At a Glance

| | |
|----------------------------|----------------------------|
| Specified in type library: | CDO.DLL |
| First available in: | CDO 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 object through the **Item** property. The Attachments collection supports the Microsoft® Visual Basic® **For Each** statement. For more information on collections, see [Object Collections](#).

Properties

| Name | Available since 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 since version | Parameters |
|------------------------|-------------------------|--|
| Add | 1.0.a | (optional) <i>name</i> as String ,
(optional) <i>position</i> as Long ,
(optional) <i>type</i> as Long ,
(optional) <i>source</i> as String |
| Delete | 1.0.a | (none) |

Add Method (Attachments Collection)

Group

The **Add** method creates and returns a new [Attachment](#) object in the Attachments collection.

Syntax

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

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 **CdoFileData**, **CdoFileLink**, **CdoOLE**, or **CdoEmbeddedMessage**. The default value is **CdoFileData**.

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 <i>type</i> parameter | Value of <i>source</i> parameter |
|--------------------------------|---|
| CdoFileData | 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. |
| CdoFileLink | 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. |
| CdoOLE | 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. |
| CdoEmbeddedMessage | Specifies the ID property of the message to |

be embedded. The message is copied into the attachment.

When the *type* parameter has the value **CdoFileLink**, 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* **CdoFileLink**, CDO does not validate the file name.

If you do not specify the *name*, *type*, and *source* parameters when you call the **Add** method, you must later explicitly set these properties. In this case you must set **Type** before you set **Source**. Failure to do this can result in a return of **CdoE_NOT_FOUND** from the **ReadFromFile** or **WriteToFile** method. If the type is **CdoOLE**, you must also call **ReadFromFile** on the new Attachment object to load the attachment's content.

If you do not set the **Name**, **Type**, and **Source** properties on an attachment, some calendaring clients may not successfully display the AppointmentItem, MeetingItem, or Message object. Microsoft® Outlook™, for example, returns an error message when an attachment is empty, that is, has no properties set.

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

Note Microsoft Outlook supports attachments on AppointmentItem objects, but Microsoft® Schedule+ does not. For consistency of interface, the **Add** method always accepts a new attachment on an appointment, but if the appointment is associated with Schedule+, the attachment is ignored and discarded when you commit the appointment to storage with **Send** or **Update**.

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

Count Property (Attachments Collection) Group

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_actmsg  
    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)

Group

The **Delete** method removes all the Attachment objects from the Attachments collection.

Syntax

objAttachColl.Delete()

Remarks

The **Delete** method performs an irreversible operation on the collection. It calls **Release** on the collection's reference to every Attachment object. If you have another reference to an attachment, you can still access its properties and methods, but you can never again associate it with any collection because the **Add** method always creates a new object. You should **Set** your reference variable either to **Nothing** or to another attachment.

The final **Release** on each Attachment object takes place when you assign your reference variable to **Nothing**, or when you call **Delete** if you had no other reference. At this point the object is removed from memory. Attempted access to a released object results in an error return of **CdoE_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. A permanently deleted member cannot be recovered. However, the collection itself is still valid, and you can **Add** new members to it.

Item Property (Attachments Collection)

Group

The **Item** property returns a single [Attachment](#) object from the Attachments collection. Read-only.

Syntax

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

objAttachColl.**Item**(*recordKey*)

index

Long. An integer ranging from 1 to *objAttachColl*.**Count**.

recordKey

String. The MAPI record key of an individual attachment.

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

Data Type

Object (Attachment)

Remarks

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

The **Item**(*index*) syntax selects an arbitrary Attachment object within the Attachments collection.

The **Item**(*recordKey*) syntax returns the first Attachment object with a MAPI record key equivalent to the string specified by *recordKey*. You can obtain the MAPI record key of an attachment through its [Fields](#) collection, but you cannot convert it programmatically to the requisite string format with Visual Basic. The record key is in a counted binary format for which Visual Basic and CDO have no corresponding data type.

The *recordKey* parameter corresponds to the MAPI property PR_RECORD_KEY, converted to a string of hexadecimal characters.

Although the **Item** property itself is read-only, the [Attachment](#) object it returns can be accessed in the normal manner, and its properties retain their respective read/write or read-only accessibility.

Example

This code fragment shows the [Count](#) and **Item** properties working together to traverse the collection:

```
' from Util_SmallCollectionCount(objAttachColl As Object)
Dim strItemName(100) as String
Dim i As Integer ' loop counter
' error handling omitted from this fragment ...
For i = 1 To objAttachColl.Count Step 1
    strItemName(i) = objAttachColl.Item(i).Name
    ' or = objAttachColl(i) since Item and Name are default properties
    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 a Microsoft® CDO Library object.

At a Glance

| | |
|----------------------------|-----------------------------------|
| Specified in type library: | CDO.DLL |
| First available in: | CDO Library version 1.0.a |
| Parent objects: | Fields collection |
| Child objects: | (none) |
| Default property: | <u>Value</u> |

Properties

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

Methods

| Name | Available since version | Parameters |
|----------------------------|-------------------------|----------------------------------|
| <u>Delete</u> | 1.0.a | (none) |
| <u>ReadFromFile</u> | 1.0.a | <i>fileName</i> as String |
| <u>WriteToFile</u> | 1.0.a | <i>fileName</i> as String |

Remarks

The Field object gives you the ability to access MAPI properties on an [AddressEntry](#), [AddressEntryFilter](#), [AddressList](#), [AppointmentItem](#), [Attachment](#), [Folder](#), [InfoStore](#), [MeetingItem](#), [Message](#), or [MessageFilter](#) object. The properties can be predefined MAPI properties or custom named properties.

You do not have to add a predefined property to the [Fields](#) collection in order to access it. You can set it or read its value by simply supplying its property tag to the collection's **Item** property. To access a custom property, you normally have to use the Fields collection's **Add** method to establish access to the property.

You can add additional properties tailored for your specific application to the [Fields](#) collection. Before adding a field for an eligible object, review the properties that are already provided by CDO. Many of the most common ones are already offered. For example, **Subject** and **Importance** are already defined as Message object properties.

Data types are preserved between MAPI properties and CDO fields, with the exception of MAPI properties of type PT_BINARY. These are converted from counted binary in MAPI to character string representation in CDO, where the characters in the string represent the hexadecimal digits of the MAPI property value. The string is converted back into counted binary when you write to the field.

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 Microsoft® Visual Basic® application as a **vbVariant** array. When you write to a multivalued field, it accepts arrays of various data types, such as **vbLong** and **vbString**. When you read from a multivalued field, however, it always returns a **vbVariant** array. You can use the **For ... Next** statement to access individual array entries, as shown in this code fragment:

```
Dim rgstr(0 To 9) As String ' field accepts a string array
' Build array of values for multivalued property
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 new field; accepts a string array
' Save/send the message, logoff, etc.

' later: code that reads the multivalued properties
Dim rgret (0 To 9) As Variant ' field returns a variant array
Set f = msg.Fields.Item("FancyName") ' Get multivalued Field
rgret = f.Value ' Get array of values into a variant array
For i = LBound(rgret) To UBound(rgret)
    MsgBox rgret(i)
Next i
```

For more information on multivalued fields, see the Field object's **Type** property.

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

Delete Method (Field Object)

Group

The **Delete** method removes the Field object from the [Fields](#) collection if the field is user-defined or optional.

Syntax

objField.Delete()

Remarks

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

The **Delete** method performs an irreversible operation on the collection. It calls **Release** on the collection's reference to the Field object. If you have another reference to the field, you can still access its properties and methods, but you can never again associate it with any collection because the **Add** method always creates a new object. You should **Set** your reference variable either to **Nothing** or to another field.

The final **Release** on the Field object takes place when you assign your reference variable to **Nothing**, or when you call **Delete** if you had no other reference. At this point the object is removed from memory. Attempted access to a released object results in an error return of **CdoE_INVALID_OBJECT**.

When you delete a member of a collection, the collection is immediately refreshed, meaning that its **Count** property is reduced by one and its members are reindexed. To access a member following the deleted member, you must use its new index value. For more information, see [Looping Through a Collection](#).

The effect of the **Delete** operation is not permanent until you use the **Update**, **Send**, or **Delete** method on the parent [AddressEntry](#), [AddressEntryFilter](#), [AddressList](#), [AppointmentItem](#), [Attachment](#), [Folder](#), [InfoStore](#), [MeetingItem](#), [Message](#), or [MessageFilter](#) object of the [Fields](#) collection.

Example

This code fragment illustrates the two situations previously explained. The **Set** statement calls **AddRef** on the first Field object. That reference survives the call to **Delete** and has to be reassigned. The second Field object is deleted without creating another reference, and no other action is necessary.

```
' assume valid Message object
Set objField = objMessage.Fields.Item(1)
objField.Delete ' still have a reference from Set statement
' ... other operations on objField possible but pointless ...
Set objField = Nothing ' necessary to remove reference
' ...
objMessage.Fields.Item(2).Delete ' no reference to remove
```

ID Property (Field Object)

Group

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's **ID** property is unique among identifier properties supported in CDO. 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 CDO **Type** property. There is a correspondence between the two entities, but their value sets are not the same. The Field object's **ID** property contains the MAPI property type; its **Type** property contains the Visual Basic® data type used by CDO.

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: &H" _
    & Hex(objOneField.ID)
```

Index Property (Field Object)

Group

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

Syntax

objField.Index

Data Type

Long

Remarks

The **Index** property indicates this object's position within the parent Fields collection. It can be saved and used later with the collection's **Item** property to reselect the same field in the collection.

The first object in the collection has an **Index** value of 1.

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)

Group

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

Syntax

objField.Name

objField.Name(*PropsetID*)

objField

Object. The Field object.

PropsetID

Optional. String. Contains the **GUID** that identifies the property set, represented as a string of hexadecimal characters. When this identifier is not present, the property is created within the default property set. The default property set is either the property set most recently supplied to the **SetNamespace** method, or the initial default property set value, PS_PUBLIC_STRINGS.

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.

The length of the name of a custom property is limited to 120 characters. An attempt to exceed this limit returns an error. Note that this limitation does not apply to the value of the property.

Field objects used to access predefined MAPI properties do not have names. Names 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)

Group

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

Syntax

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

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.

ReadFromFile is not supported for simple types, such as **vbInteger**, **vbLong**, and **vbBoolean**. Microsoft® Visual Basic® provides common functions to read and write these base types to and from files. The **ReadFromFile** method fails unless the **Type** property of the Field object is **vbString** or **vbBlob**.

MAPI properties of type PT_BINARY are read from persistent storage in counted binary format but 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 **CdoE_INTERFACE_NOT_SUPPORTED** on Field objects obtained from a Folder object's Fields collection. It also returns **CdoE_INTERFACE_NOT_SUPPORTED** on fields from an AppointmentItem object's Fields collection if the AppointmentItem represents an instantiated individual recurrence of a recurring appointment.

See Also

[WriteToFile Method \(Field Object\)](#)

Type Property (Field Object)

Group

The **Type** property returns the data type of the Field object. Read-only.

Syntax

objField.Type

Data Type

Integer

Remarks

The **Type** property contains the data type of the Field object and determines the range of valid values that can be supplied for the **Value** property. You set the **Type** property when you first create the field by setting the *Class* parameter of the *Fields* collection's **Add** method. After that, you cannot change the **Type** property.

Valid data types are as follows:

| Type | Description | Decimal value | OLE variant type | MAPI property type |
|---------------------|---|---------------|------------------|--------------------|
| vbArray | Multivalued type | 8192 | VT_ARRAY | PT_MV_FLAG |
| vbBlob | Binary (unknown format) | 65 | VT_BLOB | PT_BINARY |
| vbBoolean | Boolean | 11 | VT_BOOL | PT_BOOLEAN |
| vbCurrency | 8-byte integer (scaled by 10000) | 6 | VT_CY | PT_CURRENCY |
| vbDataObject | Data object | 13 | VT_UNKNOWN | PT_OBJECT |
| vbDate | 8-byte real (date in integer, time in fraction) | 7 | VT_DATE | PT_APPTIME |
| vbDouble | 8-byte real (floating point) | 5 | VT_R8 | PT_DOUBLE, PT_R8 |
| vbEmpty | Not initialized | 0 | VT_DEREF | PT_UNSPECIFIED |
| vbInteger | 2-byte integer | 2 | VT_I2 | PT_I2, PT_SHORT |
| vbLong | 4-byte integer | 3 | VT_I4 | PT_I4, PT_LONG |
| vbNull | Null (no | 1 | VT_NULL | PT_NULL |

| | | | | |
|------------------|----------------------------------|----|---------|-----------------|
| | valid data) | | | |
| vbSingle | 4-byte real (floating point) | 4 | VT_R4 | PT_FLOAT, PT_R4 |
| vbString | String | 8 | VT_BSTR | PT_TSTRING |
| vbVariant | Variant (object of unknown type) | 12 | | PT_UNSPECIFIED |

The current version of CDO does not support the **vbNull** and **vbDataObject** data types. The **vbEmpty** data type should never appear as the value of the **Type** property because the **Add** method should derive the data type from the new field's value if the *Class* parameter is set to **vbEmpty**.

The **vbArray** data type must always be used in conjunction with one of the other types, for example **vbArray + vbString**. The entire array must be of the same data type. Note that operations such as comparison cannot be done with a single operator on types involving **vbArray**.

When you use a multivalued field in conjunction with an array, you must declare the array to be of the appropriate data type. The data type depends on whether you are writing to the field or reading from it, because a multivalued field accepts arrays of various data types but always returns a **vbVariant** array. To write to the field, that is, to copy an array to it, declare the type of the array the same as the base type of the field:

```
Dim Codes(10) As Integer ' NOT just Dim Codes(10)
' ...
Set objCodesField = objFieldsColl.Add("Codes", vbArray + vbInteger)
objCodesField.Value = Codes
```

Failure to do this results in a **CdoE_INVALID_TYPE** error. To read from the field, that is, to copy it to an array, declare the array as **Variant**:

```
Dim Tags(10) As Variant ' NOT Dim Tags(10) or Dim Tags (10) As Long
' ... instantiate object objSource exposing objTagsField ...
Set objTagsField = objSource.Fields.Item("Tags")
objTagsField.Value = Tags
```

Failure to do this results in a VB compiler error.

MAPI stores all custom properties that represent date and time information using Greenwich Mean Time (GMT), also known as Coordinated Universal Time (UTC). CDO 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)

Group

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

Syntax

objField.Value

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

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.

When you use a multivalued type, that is, a field including **vbArray** in its data type, you can access its individual elements, but you cannot subscript the field's **Value** property directly. You must copy the field's contents to a declared array variable for this purpose:

```
Dim colFields As Fields ' collection of fields on some object
' assume colFields and its parent are valid ...
Dim objField As Field ' single field, to be made multivalued string
Dim StringValues (100) As String ' to hold field's multiple values
' ...
arrStrings = Array("String1", "String2", "String3")
Set objField = colFields.Add("Strings", vbArray+vbString, arrStrings)
' ...
StringValues = objField.Value ' copy to array variable
MsgBox "Count: " & UBound(objField.Value) - LBound(objField.Value) + 1
For i = LBound(objField.Value) To UBound(objField.Value)
    MsgBox "Value: " & StringValues(i)
Next i
' alternate loop without subscripting:
'   For Each Value In objField.Value
'       MsgBox "Value: " & Value
'   Next
```

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
'           or just objOneField since .Value is default property
```


WriteToFile Method (Field Object)

Group

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

Syntax

objField.WriteToFile(*fileName*)

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.

WriteToFile is not supported for simple types, such as **vbInteger**, **vbLong**, and **vbBoolean**. Microsoft® Visual Basic® provides common functions to read and write these base types to and from files. The **WriteToFile** method fails unless the **Type** property of the Field object is **vbString** or **vbBlob**.

MAPI properties of type PT_BINARY are represented in a hexadecimal string format by CDO but written to persistent storage in counted 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.

WriteFromFile returns **CdoE_INTERFACE_NOT_SUPPORTED** on Field objects obtained from a Folder object's Fields collection. It also returns **CdoE_INTERFACE_NOT_SUPPORTED** on fields from an AppointmentItem object's Fields collection if the AppointmentItem represents an instantiated individual recurrence of a recurring appointment.

See Also

[ReadFromFile Method \(Field Object\)](#)

Fields Collection Object

The Fields collection object contains one or more Field objects.

At a Glance

| | |
|----------------------------|--|
| Specified in type library: | CDO.DLL |
| First available in: | CDO Library version 1.0.a |
| Parent objects: | AddressEntry
AddressEntryFilter
AddressList
AppointmentItem
Attachment
Folder
InfoStore
MeetingItem
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 Microsoft® Visual Basic® **For Each** statement. For more information on collections, see [Object Collections](#).

Properties

| Name | Available since 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, AddressEntryFilter object, AddressList object, AppointmentItem object, Attachment object, Folder object, InfoStore object, MeetingItem object, Message object, or MessageFilter object | Read-only |
| Session | 1.0.a | Session object | Read-only |

Methods

| Name | Available since version | Parameters |
|------|-------------------------|------------|
|------|-------------------------|------------|

| | | |
|----------------------------|-------|--|
| <u>Add</u> | 1.0.a | <i>name</i> as String ,
<i>Class</i> as Long ,
<i>value</i> as Variant ,
(optional) <i>PropsetID</i> as String ,
<i>PropTag</i> as Long |
| <u>Delete</u> | 1.0.a | (none) |
| <u>SetNamespace</u> | 1.0.a | <i>PropsetID</i> 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 &H8000. These are known as *unnamed properties* because they are 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.

Data types are preserved between MAPI properties and CDO fields, with the exception of MAPI properties of type PT_BINARY. These are converted from counted binary in MAPI to character string representation in CDO, where the characters in the string represent the hexadecimal digits of the MAPI property value. The string is converted back into counted binary when you write to the field.

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 &H8000 by CDO, 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 corresponds in the context of CDO to a *name space*.

A property set is defined by a **GUID**, or globally unique identifier. CDO 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 CDO Library.

You can also choose to organize your custom properties within their own name space by defining your own property set. The **Add** and **Item** properties and the **SetNamespace** method 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 &H8000). |
| PS_PUBLIC_STRINGS | Default property set for custom properties added using CDO. |
| 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_ADDRESS
ES | 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 IResult As Long
Dim IGuid As GUID

' call CoCreateGuid, then convert the result to a hex string
IResult = CoCreateGuid(IGuid)
If IResult = S_OK Then
    strPropID = Hex$(IGuid.Guid1) & Hex$(IGuid.Guid2)
    strPropID = strPropID & Hex$(IGuid.Guid3)
    strPropID = strPropID & Hex$(IGuid.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 "COM and ActiveX Object Services" in the Microsoft Platform SDK documentation.

MAPI stores all custom properties that represent date and time information using Greenwich Mean Time (GMT). CDO 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, or the MAPI property tag:

```
Set objNamedField = objFolder.Fields.Item("BalanceDue")
Set objNamedField2 = objMessage.Fields.Item("Keyword")
Set objIndexedField = objMessage.Fields.Item(3)
propTag = &H0E180003 ' VB4.0: propTag = CdoPR_MESSAGE_DOWNLOAD_TIME
Set objMAPIField = objMessage.Fields.Item(propTag)
```

Add Method (Fields Collection)

Group

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)

objField

On successful return, contains the new Field object.

objFieldsColl

Required. The Fields collection object.

name

Required. String. The property name assigned to a custom named property.

Class

Required. Long. 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 |
|---------------------|---|---------------|------------------|
| vbArray | Multivalued type | 8192 | VT_ARRAY |
| vbBlob | Binary (unknown format) | 65 | VT_BLOB |
| vbBoolean | Boolean | 11 | VT_BOOL |
| vbCurrency | 8-byte integer (scaled by 10000) | 6 | VT_CY |
| vbDataObject | Data object | 13 | VT_UNKNOWN |
| vbDate | 8-byte real (date in integer, time in fraction) | 7 | VT_DATE |
| vbDouble | 8-byte real (floating point) | 5 | VT_R8 |
| vbEmpty | Not initialized | 0 | VT_DEREF |
| vbInteger | 2-byte integer | 2 | VT_I2 |
| vbLong | 4-byte integer | 3 | VT_I4 |
| vbNull | Null (no valid data) | 1 | VT_NULL |
| vbSingle | 4-byte real (floating point) | 4 | VT_R4 |
| vbString | String | 8 | VT_BSTR |
| vbVariant | Variant (object of unknown type) | 12 | |

The current version of CDO does not support the **vbNull** and **vbDataObject** data types.

value

Required (optional in first syntax). Variant. The value of the field, of the data type specified in the *Class* parameter or implicit in the *PropTag* parameter. You can change the value later by setting it directly or by subsequent calls to the Field object's **ReadFromFile** method.

PropsetID

Optional. String. Contains the **GUID** that identifies the property set, represented as a string of hexadecimal characters. When this identifier is not present, the property is created within the default

property set. The default property set is either the property set most recently supplied to the **SetNamespace** method, or the initial default property set value, PS_PUBLIC_STRINGS.

PropTag

Required. Long. The MAPI property tag for a predefined MAPI property.

Remarks

The **Field** object created by the **Add** method always represents a MAPI property. This can be either a predefined MAPI property, which is designated by a property identifier, or a custom property, which is designated by a unique name that MAPI associates with an identifier by means of a name-identifier mapping. This mapping makes use of the property set **GUID** that is common to every named property in that property set.

The first syntax is used for a named property. The *name* parameter contains the custom name that MAPI maps to a property identifier. You can optionally include the property set **GUID** with the name as an alternative to using the *PropsetID* parameter. If you elect this option, the **GUID** is placed in braces immediately preceding the property name itself. If the property set **GUID** is supplied in both the *name* and *PropsetID* parameters, the value in *PropsetID* takes precedence.

The length of the name of a custom property is limited to 120 characters. An attempt to exceed this limit returns an error. Note that this limitation does not apply to the value of the property.

If the *Class* parameter contains **vbEmpty** or an invalid setting, the **Add** method attempts to derive the data type from the new field's value. If this attempt fails, for example if the *value* parameter is not set, the **Add** method returns **CdoE_NO_SUPPORT**.

Note The Microsoft® Exchange Server supports only predefined MAPI properties on **Folder** objects. You cannot add or define custom named properties on Microsoft Exchange Server folders. Other servers may support named properties on folders, however.

The second syntax is used for a predefined MAPI property. The *PropTag* parameter 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.

Support for the **Add** method is provider-dependent. Not all providers support named properties.

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

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**. The entire array must be of the same data type. Note that operations such as comparison cannot be done with a single operator on types involving **vbArray**.

When you use a multivalued field in conjunction with an array, you must declare the array to be of the appropriate data type. The data type depends on whether you are writing to the field or reading from it, because a multivalued field accepts arrays of various data types but always returns a **vbVariant** array. To write to the field, that is, to copy an array to it, declare the type of the array the same as the base type of the field:

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

Failure to do this results in a **CdoE_INVALID_TYPE** error. To read from the field, that is, to copy it to an array, declare the array as **Variant**:

```
Dim Tags(10) As Variant ' NOT Dim Tags(10) or Dim Tags (10) As Long
' ... instantiate object objSource exposing objTagsField ...
Set objTagsField = objSource.Fields.Item("Tags")
objTagsField.Value = Tags
```

Failure to do this results in a VB compiler error.

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).

MAPI stores all custom properties that represent date and time information using Greenwich Mean Time (GMT). CDO 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 integer type value
  MsgBox "Field type = " & objOneField.Type
```

Count Property (Fields Collection)

Group

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)

Group

The **Delete** method removes all user-defined and optional Field objects from the Fields collection.

Syntax

objFieldsColl.Delete()

Remarks

The **Delete** method operates only on user-defined fields and on fields considered optional by the underlying provider.

The **Delete** method performs an irreversible operation on the collection. It calls **Release** on the collection's reference to every Field object. If you have another reference to a field, you can still access its properties and methods, but you can never again associate it with any collection because the **Add** method always creates a new object. You should **Set** your reference variable either to **Nothing** or to another field.

The final **Release** on each Field object takes place when you assign your reference variable to **Nothing**, or when you call **Delete** if you had no other reference. At this point the object is removed from memory. Attempted access to a released object results in an error return of **CdoE_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. A permanently deleted member cannot be recovered. However, the collection itself is still valid, and you can Add new members to it.

Item Property (Fields Collection)

Group

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

Integer. Value must be greater than 0 and less than or equal to 65,535 (&HFFFF). Specifies the index within the Fields collection.

proptag

Long. Value must be greater than or equal to 65,536 (&H10000). Specifies the property tag value for the MAPI property to be retrieved.

name

String. Contains the custom name of the user-defined property, or a string representation of the of its property tag.

propsetID

Optional. String. Contains the **GUID** that identifies 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 most recently set by this collection's [SetNamespace](#) method, or the initial default property set value, PS_PUBLIC_STRINGS.

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

Data Type

Object (Field)

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.

Note that not all MAPI property types can be manipulated within CDO. In particular, the MAPI types PT_ERROR, PT_I8, PT_LONGLONG, and PT_SYSTIME do not have corresponding CDO data types. Properties of these types can be obtained with the **Item** property but cannot be assigned, altered, or compared within CDO. They can, however, be rendered by the [ContainerRenderer](#) object's [RenderProperty](#) method or the [ObjectRenderer](#) object's [RenderProperty](#) method.

The *name* parameter in the third syntax must be a string. It contains either the custom property's name or its property tag. The tag must be represented as an ASCII string, which must consist of the characters "0x" followed by up to eight hexadecimal digits. Combined with the **GUID** in the *propsetID* parameter, this syntax allows you to access properties from a property set other than your default set, either by name or by property tag.

If you have a custom property name that starts with the string "0x" you cannot access it with the *name*

parameter, because the third syntax attempts to interpret the characters following "0x" as hexadecimal digits.

If the specified property is not present in the Fields collection, the **Item** property returns **CdoE_NOT_FOUND**.

Several macros for C/C++ programmers are available in the *MAPI Programmer's Reference* to help manipulate the MAPI property tag data structure. The macros **PROP_TYPE** and **PROP_ID** extract the property type and property identifier from the property tag. The macro **PROP_TAG** builds the property tag from the type and identifier components.

Although the **Item** property itself is read-only, the Field object it returns can be accessed in the normal manner, and its properties retain their respective read/write or read-only accessibility.

Example

This code fragment accesses 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")
' could be objFieldsColl("Keyword") since .Item is default property
If objOneField Is Nothing Then
    MsgBox "could not select Field object"
    Exit Function
End If
If "" = objOneField.Name Then
    MsgBox "Keyword has no name; ID = " & objOneField.ID
Else
    MsgBox "Keyword name = " & objOneField.Name
End If
```

You can also use the **Item** property to access MAPI properties. The defined MAPI properties are unnamed properties and can only be accessed using the numeric *proptag* value. They cannot be accessed using a string that represents the name. This 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
IValue = CdoPR_MESSAGE_CLASS ' = &H001A001E
' high-order 16 bits are property ID; low-order are property type
Set objOneField = objFieldsColl.Item(IValue)
If objOneField Is Nothing Then
    MsgBox "Could not get the Field using the value " & IValue
    Exit Function
Else
    strMsg = "Used " & IValue _
            & " to access the MAPI property " _
            & "PR_MESSAGE_CLASS: type = " _
            & objOneField.Type _
            & "; value = " _
            & objOneField.Value
```

```
MsgBox strMsg  
End If
```

CDO 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.

For more information on working with MAPI properties, see [Customizing a Folder or Message and Viewing MAPI Properties](#).

SetNamespace Method (Fields Collection) Group

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

Syntax

objFieldsColl.**SetNamespace** (*PropsetID*)

objFieldsColl

Required. The Fields collection object.

PropsetID

Required. String. Contains the **GUID** that uniquely identifies the property set, represented as a string of hexadecimal characters. This becomes the default property set to be used in subsequent named property accesses to a Field object in this Fields collection. An empty string resets the default property set to PS_PUBLIC_STRINGS.

Remarks

Every 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 named properties are accessed using the Fields collection's **Add** method and **Item** property.

The initial default value for every property set is PS_PUBLIC_STRINGS. To create your own property set for your named properties, supply a unique property set **GUID** to **SetNamespace**. This property set then replaces PS_PUBLIC_STRINGS as the default property set for all subsequent named property accesses in this Fields collection. The new default property set is used unless explicitly overridden by an optional *PropsetID* parameter in a property or method accessing a field.

The property set value is selected only for the current Fields collection; to use the new default property set for other collections, you must call **SetNamespace** for each collection.

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

For more information on named properties and property sets, see the *MAPI Programmer's Reference*. For more information on **UUIDGEN** and **CoCreateGuid**, see "COM and ActiveX Object Services" in the Microsoft Platform SDK documentation.

Folder Object

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

At a Glance

| | |
|----------------------------|---|
| Specified in type library: | CDO.DLL |
| First available in: | CDO Library version 1.0.a |
| Parent objects: | <u>Folders</u> collection
<u>InfoStore</u>
<u>Session</u> |
| Child objects: | <u>Fields</u> collection
<u>Folders</u> collection
<u>Messages</u> collection |
| Default property: | (none) |

Properties

| Name | Available since version | Type | Access |
|-----------------------|-------------------------|---|--|
| <u>Application</u> | 1.0.a | String | Read-only |
| <u>Class</u> | 1.0.a | Long | Read-only |
| <u>Fields</u> | 1.0.a | Field object or Fields collection object | Read-only |
| <u>FolderID</u> | 1.0.a | String | Read-only |
| <u>Folders</u> | 1.0.a | Folders collection object | Read-only |
| <u>HiddenMessages</u> | 1.2 | Messages collection object | Read-only |
| <u>ID</u> | 1.0.a | String | Read-only |
| <u>MAPIOBJECT</u> | 1.0.a | IUnknown object | Read/write
(Note: Not available to Visual Basic applications) |
| <u>Messages</u> | 1.0.a | Messages collection object | Read-only |
| <u>Name</u> | 1.0.a | String | Read/write |
| <u>Parent</u> | 1.0.a | Folders collection object or InfoStore object | Read-only |
| <u>Session</u> | 1.0.a | Session object | Read-only |
| <u>StoreID</u> | 1.0.a | String | Read-only |

Methods

| Name | Available since version | Parameters |
|------------------------|--------------------------------|---|
| <u>CopyTo</u> | 1.1 | <i>folderID</i> as String ,
(optional) <i>storeId</i> as String ,
(optional) <i>name</i> as String ,
(optional) <i>copySubfolders</i> as Boolean |
| <u>Delete</u> | 1.0.a | (none) |
| <u>IsSameAs</u> | 1.1 | <i>objFolder2</i> as Object |
| <u>MoveTo</u> | 1.1 | <i>folderID</i> as String ,
(optional) <i>storeId</i> as String |
| <u>Update</u> | 1.0.a | (optional) <i>makePermanent</i> as Boolean ,
(optional) <i>refreshObject</i> as Boolean |

Remarks

A Folder object can be rendered into HTML hypertext using the CDO Rendering ObjectRenderer object. To specify this, set the object renderer's **DataSource** property to the Folder object itself. The individual properties that can be rendered with the **RenderProperty** method are indicated in the Folder object property descriptions.

A Folder object can also be rendered as the parent of a Messages collection, using the ContainerRenderer object. The individual properties that can be rendered with the **RenderProperty** method are indicated in the Folder object property descriptions.

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

CopyTo Method (Folder Object)

Group

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

Syntax

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

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. The default value is **True**.

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)

Group

The **Delete** method removes the Folder object from its parent [Folders](#) collection or [InfoStore](#) object.

Syntax

objFolder.Delete()

Remarks

The **Delete** method performs an irreversible operation on the collection. It calls **Release** on the collection's reference to the Folder object. If you have another reference to the folder, you can still access its properties and methods, but you can never again associate it with any collection because the **Add** method always creates a new object. You should **Set** your reference variable either to **Nothing** or to another folder.

The final **Release** on the Folder object takes place when you assign your reference variable to **Nothing**, or when you call **Delete** if you had no other reference. At this point the object is removed from memory. Attempted access to a released object results in an error return of **CdoE_INVALID_OBJECT**.

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

When you delete a member of a collection, the collection is immediately refreshed, meaning that its **Count** property is reduced by one and its members are reindexed. To access a member following the deleted member, you must use its new index value. For more information, see [Looping Through a Collection](#).

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

This code fragment illustrates the two situations previously explained. The **Set** statement calls **AddRef** on the first child Folder object. That reference survives the call to **Delete** and has to be reassigned. The second child Folder object is deleted without creating another reference, and no other action is necessary.

```
' assume valid Folder object
Set objChildFolder = objFolder.Folders.Item(1)
objChildFolder.Delete ' still have a reference from Set statement
' ... other operations on objChildFolder possible but pointless ...
Set objChildFolder = Nothing ' necessary to remove reference
' ...
objFolder.Folders.Item(2).Delete ' no reference to remove
```

Fields Property (Folder Object)

Group

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

Integer. Value must be greater than 0 and less than or equal to 65,535 (&HFFFF). Specifies the index within the Fields collection.

proptag

Long. Value must be greater than or equal to 65,536 (&H10000). 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 (Field or Fields collection)

Remarks

The **Fields** property returns one or all of the fields associated with a Folder object. Each field typically corresponds to a MAPI property. Data types are preserved, except that MAPI counted binary properties are converted to and from character strings representing hexadecimal digits.

The **Fields** property provides a generic access mechanism that allows Microsoft® Visual Basic® and Microsoft® 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 **CdoPR_CONTENT_COUNT**. To access a named property, use *objFolder.Fields(name)*, where *name* is a string that represents the custom property name.

Note The Microsoft® Exchange Server supports only predefined MAPI properties on folders. You cannot add or define custom named properties on Microsoft Exchange Server folders. Other servers may support named properties on folders, however.

Although the **Fields** property itself is read-only, the collection it returns can be accessed in the normal manner through its **Add** and **Delete** methods, and the properties on its member [Field](#) objects retain their respective read/write or read-only accessibility.

The **Fields** property does not correspond to a MAPI property and cannot be rendered into HTML hypertext by the CDO Rendering Library.

Example

This code fragment displays the field name or identifier value of all [Field Object](#) objects within the Fields 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)

Group

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. However, MAPI does not require identifier values to be binary comparable. Accordingly, two identifier values can be different, yet refer to the same object. MAPI compares identifiers with the **CompareEntryIDs** method. CDO provides the **CompareIDs** method in the Session object. For more information on entry identifiers, see the *MAPI Programmer's Reference*.

A Microsoft® Schedule+ calendar folder does not have a parent folder nor reside in a message store. If you obtain the default calendar folder by passing **CdoDefaultFolderCalendar** to the Session object's **GetDefaultFolder** method, the **FolderID** property has no defined value. An attempt to access **FolderID** in this case returns **CdoE_NOT_FOUND**.

The **FolderID** property corresponds to the MAPI property PR_PARENT_ENTRYID, converted to a string of hexadecimal characters. It can be rendered into HTML hypertext using the CDO Rendering ObjectRenderer object. To specify this, set the object renderer's **DataSource** property to this Folder object and the *property* parameter of the RenderProperty method to **CdoPR_PARENT_ENTRYID**.

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 of Inbox (that is, store's root 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)

Group

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

Syntax

objFolder.Folders

Data Type

Object (Folders collection)

Remarks

Although the **Folders** property itself is read-only, the collection it returns can be accessed in the normal manner through its **Add** and **Delete** methods, and the properties on its member Folder objects retain their respective read/write or read-only accessibility.

The Microsoft® Schedule+ appointment folder is not implemented in the same way as Microsoft® Outlook™ folders and CDO folders. In particular, it does not have subfolders. An attempt to read the **Folders** property on the Schedule+ appointment folder returns **CdoE_NO_SUPPORT**.

The **Folders** property does not correspond to a MAPI property and cannot be rendered into HTML hypertext by the CDO Rendering Library.

Example

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

```
Set objFolder = objSession.Inbox
' assume objFolder is valid
If CdoFolder = objFolder.Class Then ' verify it's a Folder object
    ListFolders objFolder ' use current global folder
' *** must call ListFolders withOUT parentheses, or else VB evaluates
' *** objFolder for its default property and passes objFolder.Messages
End If

' subroutine to list folders
Sub ListFolders(objParentFolder As Folder)
Dim objFoldersColl As Folders ' the child Folders collection
Dim objOneSubfolder As Folder '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 ' there are child folders
        Set objOneSubfolder = objFoldersColl.GetFirst
        While Not objOneSubfolder Is Nothing ' loop through all children
            ListFolders objOneSubfolder ' must call without parentheses
            Set objOneSubfolder = objFoldersColl.GetNext
        Wend
    End If
Exit Sub
End If
' error handler here
End Sub
```

HiddenMessages Property (Folder Object) Group

The **HiddenMessages** property returns a Messages collection object of hidden messages in the folder. Read-only.

Syntax

objFolder.HiddenMessages

Data Type

Object (Messages collection)

Remarks

The messages in the collection returned by the **HiddenMessages** property are not visible through the Microsoft® Exchange Client, Microsoft® Outlook™, or Microsoft Outlook Web Access (OWA). These hidden messages correspond to the folder associated information (FAI) kept in a folder by MAPI.

Some messaging clients use hidden messages to accomplish special tasks, for example the processing of rules by the Microsoft Exchange Client's Inbox Assistant. You should leave the **HiddenMessages** collection unchanged unless you are familiar with the consequences of any modifications you might make.

Although the **HiddenMessages** property itself is read-only, the collection it returns can be accessed in the normal manner through its **Add** and **Delete** methods, and the properties on its member Message objects retain their respective read/write or read-only accessibility.

The **HiddenMessages** property does not correspond to a MAPI property and cannot be rendered into HTML hypertext by the CDO Rendering Library. It could be rendered as a container object by setting the ContainerRenderer object's **DataSource** property to the Messages collection object returned by the **HiddenMessages** property. The individual Message objects in the collection correspond to messages with the MSGFLAG_ASSOCIATED flag of the MAPI property PR_MESSAGE_FLAGS being set.

ID Property (Folder Object)

Group

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. However, MAPI does not require identifier values to be binary comparable. Accordingly, two identifier values can be different, yet refer to the same object. MAPI compares identifiers with the **CompareEntryIDs** method. CDO provides the **CompareIDs** method in the [Session](#) object. For more information on entry identifiers, see the *MAPI Programmer's Reference*.

The **ID** property corresponds to the MAPI property PR_ENTRYID, converted to a string of hexadecimal characters. It can be rendered into HTML hypertext using the CDO Rendering [ObjectRenderer](#) object. To specify this, set the object renderer's **DataSource** property to this Folder object and the *property* parameter of the [RenderProperty](#) method to **CdoPR_ENTRYID**.

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)

Group

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

Syntax

```
boolSame = objFolder.IsSameAs(objFolder2)
```

boolSame

On successful return, contains **True** if the two objects are the same.

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 in the underlying messaging system. 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. This is necessary because, although MAPI requires all entry identifiers to be unique, it does not require two of them identifying the same object to be identical. A generic comparison of any two objects' unique identifiers is also available with the Session object's **CompareIDs** method.

MAPIOBJECT Property (Folder Object)

Group

The **MAPIOBJECT** property returns an **IUnknown** pointer to the Folder object. Not available to Microsoft® 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 accessible only by C/C++ programs that deal with **IUnknown** objects. Visual Basic supports the **IDispatch** interface and not **IUnknown**. The **MAPIOBJECT** property is an **IUnknown** object that returns an **IMAPIFolder** interface in response to **QueryInterface**. For more information, see [Introduction to Automation](#) and [How Programmable Objects Work](#). Also see the "COM and ActiveX Object Services" section of the Microsoft Platform SDK.

If your application uses any **MAPIOBJECT** or **RawTable** properties, it must **Release** them all before calling the [Session](#) object's **Logoff** method. Failure to do so can result in unexpected behavior.

The **MAPIOBJECT** property does not correspond to a MAPI property and cannot be rendered into HTML hypertext by the CDO Rendering Library.

Messages Property (Folder Object)

Group

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

Syntax

objFolder.**Messages**

Data Type

Object (Messages collection)

Remarks

Although the **Messages** property itself is read-only, the collection it returns can be accessed in the normal manner through its **Add** and **Delete** methods, and the properties on its member Message objects retain their respective read/write or read-only accessibility.

The **Messages** property does not correspond to a MAPI property and cannot be rendered into HTML hypertext by the CDO Rendering Library. It could be rendered as a container object by setting the ContainerRenderer object's **DataSource** property to the Messages collection object returned by the **Messages** property.

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)

Group

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

Syntax

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

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)

Group

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. It can be rendered into HTML hypertext using the CDO Rendering [ObjectRenderer](#) object. To specify this, set the object renderer's **DataSource** property to this Folder object and the *property* parameter of the [RenderProperty](#) method to **CdoPR_DISPLAY_NAME**.

Example

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

StoreID Property (Folder Object)

Group

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. However, MAPI does not require identifier values to be binary comparable. Accordingly, two identifier values can be different, yet refer to the same object. MAPI compares identifiers with the **CompareEntryIDs** method. CDO provides the **CompareIDs** method in the [Session](#) object. For more information on entry identifiers, see the *MAPI Programmer's Reference*.

A Microsoft® Schedule+ calendar folder does not reside in a message store nor have a parent folder. If you obtain the default calendar folder by passing **CdoDefaultFolderCalendar** to the [Session](#) object's **GetDefaultFolder** method, the **StoreID** property has no defined value. An attempt to access **StoreID** in this case returns **CdoE_NOT_FOUND**.

The **StoreID** property corresponds to the MAPI property PR_STORE_ENTRYID, converted to a string of hexadecimal characters. It can be rendered into HTML hypertext using the CDO Rendering [ObjectRenderer](#) object. To specify this, set the object renderer's **DataSource** property to this Folder object and the *property* parameter of the [RenderProperty](#) method to **CdoPR_STORE_ENTRYID**.

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)

Group

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

Syntax

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

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, CDO 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.

At a Glance

| | |
|----------------------------|---------------------------|
| Specified in type library: | CDO.DLL |
| First available in: | CDO Library version 1.0.a |
| Parent objects: | Folder |
| Child objects: | Folder |
| Default property: | Item |

A Folders collection is considered a *large collection*, which means that the **Count** property has limited validity, and the best way to access an individual Folder object within the collection is to use either its unique identifier or the **Get** methods. For more information on collections, see [Object Collections](#).

Properties

| Name | Available since 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 | Folder object | Read-only |
| RawTable | 1.1 | IUnknown object | Read/write
(Note: Not available to Visual Basic applications) |
| Session | 1.0.a | Session object | Read-only |

Methods

| Name | Available since version | Parameters |
|-----------------------------|-------------------------|---|
| Add | 1.1 | <i>name</i> 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) <i>SortOrder</i> as Long ,
(optional) <i>PropTag</i> as Long ,
(optional) <i>PropID</i> 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**, **GetLast**, **GetNext**, 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 Microsoft® Visual Basic® **For Each** construction.

The order that items are returned by **GetFirst**, **GetLast**, **GetNext**, 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 MAPI.Session
Dim objMessage As Message
Dim objFolder As Folder

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
```


Add Method (Folders Collection)

Group

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

Syntax

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

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.

You do not need to call the **Update** method of the new Folder object when you **Add** it to the collection. However, when you set or change any of the folder's properties, you must call **Update** to save the changes in the MAPI system.

Example

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

```
Dim myInbox As Folder
Dim newFolder As Folder
Dim objSess As MAPI.Session ' use early binding for efficiency
Set objSess = CreateObject ("MAPI.Session")
Set myInbox = objSess.Inbox
' add new folder to Inbox
Set newFolder = myInbox.Folders.Add "Personal Messages"
' Update not needed until changes made
```

Count Property (Folders Collection)

Group

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

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 the value &H7FFFFFFF. Programmers needing to access individual objects in a large collection are strongly advised to use the Microsoft® 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 message store provider cannot supply the precise number of Folder objects, CDO returns &H7FFFFFFF (= $2^{31} - 1 = 2,147,483,647$) for the **Count** property. This is the largest positive value for a long integer and is intended to prevent an approximate count from prematurely terminating an indexed loop. On 32-bit platforms, this value is defined in the type library as **CdoMaxCount**. On other platforms, **CdoMaxCount** is not defined, and a program on such a platform must compare the **Count** property against &H7FFFFFFF to see if it is reliable.

If the **Count** property is not reliable, that is, if it is &H7FFFFFFF, a program using it to terminate an indexed loop must also check each returned object for a value of **Nothing** to avoid going past the end of the collection.

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 Folders ' 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)

Group

The **Delete** method removes all the Folder objects from the Folders collection.

Syntax

objFoldersColl.Delete()

Remarks

The **Delete** method performs an irreversible operation on the collection. It calls **Release** on the collection's reference to every Folder object. If you have another reference to a folder, you can still access its properties and methods, but you can never again associate it with any collection because the Add method always creates a new object. You should **Set** your reference variable either to **Nothing** or to another folder.

The final **Release** on each Folder object takes place when you assign your reference variable to **Nothing**, or when you call **Delete** if you had no other reference. At this point the object is removed from memory. Attempted access to a released object results in an error return of **CdoE_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 is still valid, and you can Add new members to it.

GetFirst Method (Folders Collection)

Group

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**()

objFolder

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

objFoldersColl

Required. The Folders collection object.

Remarks

The order that items are returned by **GetFirst**, **GetLast**, **GetNext**, 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.

GetLast Method (Folders Collection)

Group

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**()

objFolder

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

objFoldersColl

Required. The Folders collection object.

Remarks

The order that items are returned by **GetFirst**, **GetLast**, **GetNext**, 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.

GetNext Method (Folders Collection)

Group

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**()

objFolder

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

objFoldersColl

Required. The Folders collection object.

Remarks

The order that items are returned by **GetFirst**, **GetLast**, **GetNext**, 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.

If the **GetFirst** method has not been called since the Folders collection was initialized, the behavior of the **GetNext** method is not defined. This can produce unexpected results if the collection is reinitialized with a **Set** statement in every iteration of a loop. The recommended procedure is to set an explicit variable for the collection before entering the loop. For more information, see [Object Collections](#).

GetPrevious Method (Folders Collection) Group

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**()

objFolder

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

objFoldersColl

Required. The Folders collection object.

Remarks

The order that items are returned by **GetFirst**, **GetLast**, **GetNext**, 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.

If the **GetLast** method has not been called since the Folders collection was initialized, the behavior of the **GetPrevious** method is not defined. This can produce unexpected results if the collection is reinitialized with a **Set** statement in every iteration of a loop. The recommended procedure is to set an explicit variable for the collection before entering the loop. For more information, see Object Collections.

Item Property (Folders Collection)

Group

The **Item** property returns a single Folder object from the Folders collection. Read-only.

Syntax

objFoldersColl.Item(index)

objFoldersColl.Item(searchValue)

index

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

searchValue

A string used to search the Folders collection starting at the current position. The search returns the next Folder object having the current sorting property greater than or equal to the *searchValue* string.

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

Data Type

Object (Folder)

Remarks

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 returns the Folder object at the indicated position in the collection. It can be used in an indexed loop, such as the **For ... Next** construction in Visual Basic. The first item in the collection has an index of 1.

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

The **Item(searchValue)** syntax returns the next Folder object whose current sorting property is greater than or equal to the string specified by *searchValue*. This syntax starts its search at the current position.

Prefix searching is based on the current sort order of the collection. The default sort property for a Folders collection is the **Name** property of the collection's Folder objects. If you want to use the **Item(searchValue)** syntax to search the collection on another property, for example a parent folder ID, you should first call the **Sort** method specifying the **FolderID** property.

Note The **Item(searchValue)** syntax uses the **IMAPITABLE::FindRow** method, which performs a search dependent on the current sort order of the table underlying the collection. Not all tables are sorted alphabetically. The Microsoft Exchange Public Folders folder, for example, is held in a nonalphabetic order, and you should access its subfolders using the **Item(index)** syntax.

For more information on tables, bookmarks, restrictions, and sort and search orders, see the *MAPI Programmer's Reference*.

If your application is running as a Windows NT® service, you cannot access the Microsoft Exchange Public Folders through the normal hierarchy because of a notification conflict. You must use the InfoStore object's **Fields** property to obtain the Microsoft Exchange property **PR_IPM_PUBLIC_FOLDERS_ENTRYID**, property tag &H66310102. This represents the top-level public folder and allows you to access all other public folders through its **Folders** property. For more information on Windows NT services, see the Win32® Web page *Using MAPI from a Windows NT Service* at <http://www.microsoft.com/win32dev/mapi/mapiserv.htm>.

Although the **Item** property itself is read-only, the Folder object it returns can be accessed in the normal manner, and its properties retain their respective read/write or read-only accessibility.

RawTable Property (Folders Collection)

Group

The **RawTable** property returns an **IUnknown** pointer to the MAPI table object underlying the Folders collection. Not available to Microsoft® Visual Basic® applications. Read/write.

Syntax

objFoldersColl.RawTable

Data Type

Variant (**vbDataObject** format)

Remarks

The **RawTable** property is not available to Visual Basic programs. It is accessible only by C/C++ programs that deal with **IUnknown** objects. Visual Basic supports the **IDispatch** interface and not **IUnknown**. The **RawTable** property is an **IUnknown** object that returns an **IMAPITable** interface in response to **QueryInterface**. For more information, see [Introduction to Automation](#) and [How Programmable Objects Work](#). Also see the "COM and ActiveX Object Services" section of the Microsoft Platform SDK.

If your application uses any **MAPIOBJECT** or **RawTable** properties, it must **Release** them all before calling the [Session](#) object's **Logoff** method. Failure to do so can result in unexpected behavior.

The **RawTable** property does not correspond to a MAPI property and cannot be rendered into HTML hypertext by the CDO Rendering Library.

Sort Method (Folders Collection)

Group

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

Syntax

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

objFoldersColl.Sort([*SortOrder*] [, *name*])

objFoldersColl

Required. The Folders collection object.

SortOrder

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

| Value | Numeric value | Description |
|----------------------|---------------|--------------------------|
| CdoNone | 0 | No sort |
| CdoAscending | 1 | Ascending sort (default) |
| CdoDescending | 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 **CdoPR_STORE_ENTRYID**.

name

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 *name* 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 **CdoPR_DISPLAY_NAME** is used for the sort.

Each call to **Sort** generates an entirely new sort order based on the specified property. No previous sort order is retained or nested.

If the underlying messaging system does not support the sort criteria specified, for example descending order or MAPI named properties, the **Sort** method returns **CdoE_TOO_COMPLEX**.

GroupHeader Object

The GroupHeader object represents the header for a grouping of messages within a table view.

At a Glance

| | |
|----------------------------|-------------------------------------|
| Specified in type library: | CDO.DLL |
| First available in: | CDO Library version 1.1 |
| Parent objects: | Messages collection |
| Child objects: | (none) |
| Default property: | <u>Name</u> |

Properties

| Name | Available since version | Type | Access |
|---------------------------|-------------------------|----------------------------|-----------|
| <u>Application</u> | 1.1 | String | Read-only |
| <u>Class</u> | 1.1 | Long | Read-only |
| <u>Count</u> | 1.1 | Long | Read-only |
| <u>Level</u> | 1.1 | Long | Read-only |
| <u>Name</u> | 1.1 | String | Read-only |
| <u>Parent</u> | 1.1 | Messages collection object | Read-only |
| <u>Session</u> | 1.1 | Session object | Read-only |
| <u>Unread</u> | 1.1 | Long | Read-only |

Methods

(None.)

Remarks

A GroupHeader object is only instantiated when a CDO Rendering [TableView](#) is applied to a [Messages](#) collection, and furthermore only when this view is a grouped, or categorized, view. The group header indicates that the items following it in the view are grouped within its category. A grouped view is generated externally to the CDO application and cannot be created or deleted programmatically by the application. A rendering application applies the view by assigning it to the **CurrentView** property of the CDO Rendering [ContainerRenderer](#) object. Applying the grouped view causes the underlying collection to be updated with the grouping specified in the view.

Currently, categorized views are only applied to folders, and group headers can only appear in a [Messages](#) collection. Address book container views are not grouped.

Not all message store providers support group headers. To do so, a provider must be able to apply a MAPI restriction to the **IMAPITable** object returned by the **RawTable** property of the collection being viewed. The private and public message stores included with the Microsoft® Exchange Server support group headers.

Group headers are nonpersistent objects instantiated automatically to represent the categorization rows in a grouped view. They are not stored anywhere. Group headers, like grouped views, cannot be created or deleted programmatically by a CDO application. The only way to create GroupHeader objects is to apply a grouped view to a rendering object. The grouping is specified externally as part of the view, for example by the Microsoft® Exchange Client. A group header is released, with no

persistent copy, when its Messages collection is released.

A GroupHeader object corresponds to a categorization row in a MAPI view table. The group header is only needed when you traverse a Messages collection and need to distinguish between categorization and message items. When you render a categorized table with the CDO Rendering ContainerRenderer object, the categorization rows are rendered as well as the messages.

GroupHeader objects are included with Message objects in the **Count** property of the Messages collection and are accessible through its **Item** property. They can also be returned by the Messages collection's **Get** methods.

Count Property (GroupHeader Object)

Group

The **Count** property returns the total number of items in the group. Read-only.

Syntax

objGroupHdr.Count

Data Type

Long

Remarks

The **Count** property represents the number of [Message](#) objects that are grouped under this group header. It includes both read and unread messages. If -1 is returned, an accurate count is not available.

The **Count** property corresponds to the MAPI property PR_CONTENT_COUNT.

Level Property (GroupHeader Object)

Group

The **Level** property returns the indentation level of the group header within the table view. Read-only.

Syntax

objGroupHdr.Level

Data Type

Long

Remarks

The **Level** property represents the nesting depth of this group header within the table view. The top-level group is at level 1, the outermost level. The maximum permitted grouping depth is 4.

The number of categories, or levels of grouping, in a particular table view is given by the **Categories** property of the CDO Rendering TableView object.

The **Level** property corresponds to the MAPI property PR_DEPTH, incremented by a value of 1.

Name Property (GroupHeader Object)

Group

The **Name** property returns a string that can be used as text for various categories in the table view. Read-only.

Syntax

objGroupHdr.Name

The **Name** property is the default property of a GroupHeader object, meaning that *objGroupHdr* is syntactically equivalent to *objGroupHdr.Name* in Microsoft® Visual Basic® code.

Data Type

String

Remarks

The **Name** property returns text that can be used for view categories such as sender, recipient, subject, conversation topic, keyword, size, time sent, and time received.

A rendering object such as the CDO Rendering ContainerRenderer object should use the string in the **Name** property to render group headers. The string should be followed by the message counts in the **Count** and **Unread** properties, for example:

<category header string> (5 items, 3 unread)

or, if all items are marked as having been read:

<category header string> (5 items)

Unread Property (GroupHeader Object)

Group

The **Unread** property returns the number of unread messages in the group. Read-only.

Syntax

objGroupHdr.Unread

Data Type

Long

Remarks

The **Unread** property represents the number of [Message](#) objects grouped under this group header that have not been marked as read. If -1 is returned, an accurate count is not available.

The **Unread** property corresponds to the MAPI property PR_CONTENT_UNREAD.

InfoStore Object

The InfoStore object provides access to the folder hierarchy of a message store.

At a Glance

| | |
|----------------------------|--|
| Specified in type library: | CDO.DLL |
| First available in: | CDO Library version 1.0.a |
| Parent objects: | Fields collection
InfoStores collection |
| Child objects: | Folder |
| Default property: | Name |

Properties

| Name | Available since 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 |
| ID | 1.0.a | String | Read-only |
| Index | 1.0.a | Long | Read-only |
| MAPIOBJECT | 1.1 | IUnknown object | Read/write
(Note: Not available to Visual Basic applications) |
| 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 since version | Parameters |
|--------------------------|-------------------------|---------------------------------------|
| IsSameAs | 1.1 | <i>objInfoStore2</i> 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)
If objInfoStore is Nothing Then
    MsgBox "Cannot open session's first message store"
    ' Exit ...
End If
Set objIPMRoot = objInfoStore.RootFolder
rootID = objIPMRoot.FolderID
Set objStoreRoot = objSession.GetInfoStore (rootID)
' ... error checking ...
```

Fields Property (InfoStore Object)

Group

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

Syntax

objInfoStore.Fields

objInfoStore.Fields(*index*)

objInfoStore.Fields(*proptag*)

objInfoStore.Fields(*name*)

index

Integer. Value must be greater than 0 and less than or equal to 65,535 (&HFFFF). Specifies the index within the Fields collection.

proptag

Long. Value must be greater than or equal to 65,536 (&H10000). 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 (Field or Fields collection)

Remarks

The **Fields** property returns one or all of the fields associated with an InfoStore object. Each field typically corresponds to a MAPI property. Data types are preserved, except that MAPI counted binary properties are converted to and from character strings representing hexadecimal digits.

The **Fields** property provides a generic access mechanism that allows Microsoft® Visual Basic® and Microsoft® 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 *objInfoStore.Fields(proptag)*, where *proptag* is the 32-bit MAPI property tag associated with the property, such as **CdoPR_STORE_SUPPORT_MASK**. To access a named property, use *objInfoStore.Fields(name)*, where *name* is a string that represents the custom property name.

Although the **Fields** property itself is read-only, the collection it returns can be accessed in the normal manner through its **Add** and **Delete** methods, and the properties on its member Field objects retain their respective read/write or read-only accessibility.

The **Fields(name)** syntax is not supported by all message store providers. An InfoStore that does not support named properties returns **CdoE_NO_SUPPORT** for this syntax.

If your application is running as a Microsoft Windows NT® service, you cannot access the Microsoft Exchange Public Folders through the normal hierarchy because of a notification conflict. You must use the InfoStore's **Fields** property to obtain the Microsoft Exchange property **PR_IPM_PUBLIC_FOLDERS_ENTRYID**, property tag &H66310102. This represents the top-level public folder and allows you to access all other public folders through its **Folders** property. For more information on Windows NT services, see the Win32® Web page *Using MAPI from a Windows NT Service* at <http://www.microsoft.com/win32dev/mapi/mapiserv.htm>.

Example

This code fragment accesses the root of the Public Folders subtree of a message store:

```
Dim objSess As Session ' assume logged on to valid session
Dim objInfoStore As InfoStore ' assume opened and valid
```

```
Dim strPFRootID As String ' binary entry ID returned as hex string
Dim objPFRoot As Folder ' root folder of Public Folders
tagPFRootID = &H66310102 ' PR_IPM_PUBLIC_FOLDERS_ENTRYID
strPFRootID = objInfoStore.Fields(tagPFRootID) ' entry ID
MsgBox "Public Folders root folder ID = " & strPFRootID
Set objPFRoot = objSession.GetFolder(strPFRootID)
```

ID Property (InfoStore Object)

Group

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 InfoStore ' 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)

Group

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 saved and used later with the collection's [Item](#) property to reselect the same message store in the collection.

The first object in the collection has an **Index** value of 1.

Example

```
Function InfoStoresGetByIndex()
Dim lIndex As Long
Dim objOneInfoStore As InfoStore ' 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)

Group

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

Syntax

```
boolSame = objInfoStore.IsSameAs(objInfoStore2)
```

boolSame

On successful return, contains **True** if the two objects are the same.

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 in the underlying messaging system. 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. This is necessary because, although MAPI requires all entry identifiers to be unique, it does not require two of them identifying the same object to be identical. A generic comparison of any two objects' unique identifiers is also available with the Session object's **CompareIDs** method.

MAPIOBJECT Property (InfoStore Object)

Group

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

Syntax

objInfoStore.**MAPIOBJECT**

Data Type

Variant (**vbDataObject** format)

Remarks

The **MAPIOBJECT** property is not available to Visual Basic programs. It is accessible only by C/C++ programs that deal with **IUnknown** objects. Visual Basic supports the **IDispatch** interface and not **IUnknown**. The **MAPIOBJECT** property is an **IUnknown** object that returns an **IMsgStore** interface in response to **QueryInterface**. For more information, see [Introduction to Automation](#) and [How Programmable Objects Work](#). Also see the "COM and ActiveX Object Services" section of the Microsoft Platform SDK.

If your application uses any **MAPIOBJECT** or **RawTable** properties, it must **Release** them all before calling the [Session](#) object's **Logoff** method. Failure to do so can result in unexpected behavior.

The **MAPIOBJECT** property does not correspond to a MAPI property and cannot be rendered into HTML hypertext by the CDO Rendering Library.

Name Property (InfoStore Object)

Group

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

Syntax

objInfoStore.Name

The **Name** property is the default property of an InfoStore object, meaning that *objInfoStore* is syntactically equivalent to *objInfoStore.Name* in Microsoft® Visual Basic® code.

Data Type

String

Remarks

The **Name** property can be specified as the parameter to the **Item** property of the InfoStores collection if you know the name of the message store.

The string "Public Folders" is typically the name of the InfoStore object that contains the public folders.

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

Example

```
Dim objInfoStore As InfoStore ' assume valid InfoStore object
MsgBox "InfoStore name = " & objInfoStore.Name
```

ProviderName Property (InfoStore Object) Group

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 accessible as a CDO Library InfoStore object.

The **ProviderName** property corresponds to the MAPI property PR_PROVIDER_DISPLAY.

Example

```
Dim objInfoStore As InfoStore ' assume valid InfoStore object
MsgBox "Message store provider name = " & objInfoStore.ProviderName
```

RootFolder Property (InfoStore Object)

Group

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)

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, CDO 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.

If your application is running as a Microsoft® Windows NT® service, you cannot access the Microsoft Exchange Public Folders through the normal hierarchy because of a notification conflict. You must use the InfoStore's **Fields** property to obtain the Microsoft Exchange property PR_IPM_PUBLIC_FOLDERS_ENTRYID, property tag &H66310102. This represents the top-level public folder and allows you to access all other public folders through its **Folders** property. For more information on Windows NT services, see the Win32® Web page *Using MAPI from a Windows NT Service* at <http://www.microsoft.com/win32dev/mapi/mapiserv.htm>.

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.

At a Glance

| | |
|----------------------------|---------------------------|
| Specified in type library: | CDO.DLL |
| First available in: | CDO 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 Microsoft® Visual Basic® **For Each** statement. For more information on collections, see [Object Collections](#).

Properties

| Name | Available since 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 public and private folders.

CDO 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)

Group

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)

Group

The **Item** property returns a single [InfoStore](#) object from the InfoStores collection. Read-only.

Syntax

objInfoStoresColl.**Item**(*index*)

objInfoStoresColl.**Item**(*storeName*)

index

A long integer ranging from 1 to *objInfoStoresColl*.**Count**.

storeName

A string representing the name of the desired InfoStore. This can be obtained from the **Name** property of an InfoStore object.

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

Data Type

Object (InfoStore)

Remarks

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

Although the **Item** property itself is read-only, the [InfoStore](#) object it returns can be accessed in the normal manner, and its properties retain their respective read/write or read-only accessibility.

Example

For more information on using the **Count** and **Item** properties in an InfoStores collection, see the example in the [Count](#) property.

MeetingItem Object

The MeetingItem object represents a meeting in a folder.

At a Glance

| | |
|----------------------------|--|
| Specified in type library: | CDO.DLL |
| First available in: | CDO Library version 1.2 |
| Parent objects: | Messages collection |
| Child objects: | Attachments collection
Fields collection
Recipients collection |
| Default property: | Subject |

The MeetingItem object is a subclass of the [Message](#) object and exposes all the same properties and methods. In the following tables of properties and methods, those that are common with the Message object are linked to their descriptions for the Message object. Only the properties and methods unique to the MeetingItem object are described in this section.

Properties

| Name | Available since version | Type | Access |
|-----------------------------------|-------------------------|--|--|
| Application | 1.2 | String | Read-only |
| Attachments | 1.2 | Attachment object or Attachments collection object | Read-only |
| Categories | 1.2 | String array | Read/write |
| Class | 1.2 | Long | Read-only |
| Conversation | 1.2 | (Obsolete. Do not use.) | Read/write |
| ConversationIndex | 1.2 | String | Read/write |
| ConversationTopic | 1.2 | String | Read/write |
| DeliveryReceipt | 1.2 | Boolean | Read/write |
| Encrypted | 1.2 | Boolean | Read/write |
| Fields | 1.2 | Field object or Fields collection object | Read-only |
| FolderID | 1.2 | String | Read-only |
| ID | 1.2 | String | Read-only |
| Importance | 1.2 | Long | Read/write |
| MAPIOBJECT | 1.2 | IUnknown object | Read/write
(Note: Not available to Visual Basic applications) |

| | | | |
|--------------------------------|-----|---|------------|
| <u>MeetingType</u> | 1.2 | Long | Read-only |
| <u>Parent</u> | 1.2 | Messages collection object | Read-only |
| <u>ReadReceipt</u> | 1.2 | Boolean | Read/write |
| <u>Recipients</u> | 1.2 | Recipients object or Recipients collection object | Read/write |
| <u>Sender</u> | 1.2 | AddressEntry object | Read/write |
| <u>Sensitivity</u> | 1.2 | Long | Read/write |
| <u>Sent</u> | 1.2 | Boolean | Read/write |
| <u>Session</u> | 1.2 | Session object | Read-only |
| <u>Signed</u> | 1.2 | Boolean | Read/write |
| <u>Size</u> | 1.2 | Long | Read-only |
| <u>StoreID</u> | 1.2 | String | Read-only |
| <u>Subject</u> | 1.2 | String | Read/write |
| <u>Submitted</u> | 1.2 | Boolean | Read/write |
| <u>Text</u> | 1.2 | String | Read/write |
| <u>TimeCreated</u> | 1.2 | Variant (vbDate format) | Read-only |
| <u>TimeExpired</u> | 1.2 | Variant (vbDate format) | Read/write |
| <u>TimeLastModified</u> | 1.2 | Variant (vbDate format) | Read-only |
| <u>TimeReceived</u> | 1.2 | Variant (vbDate format) | Read/write |
| <u>TimeSent</u> | 1.2 | Variant (vbDate format) | Read/write |
| <u>Type</u> | 1.2 | String | Read/write |
| <u>Unread</u> | 1.2 | Boolean | Read/write |

Methods

| Name | Available since version | Parameters |
|--|--------------------------------|---|
| <u>CopyTo</u> | 1.2 | <i>folderID</i> as String , (optional) <i>storeID</i> as String |
| <u>Delete</u> | 1.2 | (none) |
| <u>Forward</u> | 1.2 | (none) |
| <u>GetAssociatedAppointment</u> | 1.2 | (none) |
| <u>IsSameAs</u> | 1.2 | <i>objMessage2</i> as Object |
| <u>MoveTo</u> | 1.2 | <i>folderID</i> as String , (optional) <i>storeID</i> as String |

| | | |
|------------------------|-----|---|
| <u>Options</u> | 1.2 | (optional) <i>parentWindow</i> as Long |
| <u>Reply</u> | 1.2 | (none) |
| <u>ReplyAll</u> | 1.2 | (none) |
| <u>Respond</u> | 1.2 | <i>RespondType</i> as Long |
| <u>Send</u> | 1.2 | (optional) <i>saveCopy</i> as Boolean ,
(optional) <i>showDialog</i> as Boolean ,
(optional) <i>parentWindow</i> as Long |
| <u>Update</u> | 1.2 | (optional) <i>makePermanent</i> as Boolean ,
(optional) <i>refreshObject</i> as Boolean |

Remarks

A MeetingItem object is distinguished from a Message object by its Type property containing IPM.Schedule.Meeting.Request.

New MeetingItem objects are created automatically by CDO when appointments are made into meetings. You can cause an AppointmentItem object to become a meeting by setting its MeetingStatus property to **CdoMeeting** and sending it to one or more recipients. To do this, create a Recipients collection using the appointment's Recipients property, populate the collection using its Add method, and call the appointment's Send method. CDO instantiates a MeetingItem object at the time you call **Send**. You cannot create a MeetingItem object directly.

When you turn an appointment into a meeting, you are identified as the initiating user in the appointment's Organizer property. If your calendar folder is in Microsoft® Outlook™, you also appear as one of the meeting's recipients.

A meeting item can be obtained from its parent Messages collection using the collection's Item property. To get to the Messages collection in a folder, use the Folder object's Messages property. If you know a meeting's unique identifier, you can obtain it directly from the Session object's GetMessage method.

A MeetingItem object becomes a member of the Inbox Messages collection of each recipient to which the original AppointmentItem object was sent. You can treat it programmatically like the Message objects in the collection. In particular, you can apply a MessageFilter object to the collection and filter meeting items on any properties, including the inherited Message object properties.

A recipient wishing to modify appointment information can call the MeetingItem object's GetAssociatedAppointment method to obtain an AppointmentItem object associated with the meeting request. This AppointmentItem is not the same object as the original AppointmentItem belonging to the meeting's organizer. It has the same property settings when it is first instantiated, but the recipient can change them as desired to make it different from the original. This AppointmentItem object is saved in the recipient's active calendar folder when its Update method is called.

To accept a meeting request, a recipient calls the MeetingItem object's Respond method with the *RespondType* parameter set to **CdoResponseAccepted**. **Respond** returns another MeetingItem object with the MeetingType property set to **CdoMeetingResponse**, which the recipient can Send back to the organizer. Calling **Respond** with either **CdoResponseAccepted** or **CdoResponseTentative** causes an associated AppointmentItem object to be created and saved in the recipient's active calendar folder.

To decline the meeting request, a recipient calls **Respond** with *RespondType* set to **CdoResponseDeclined**. No AppointmentItem object is created in the recipient's calendar folder in this case. However, any existing AppointmentItem object is left undeleted in the folder. An associated AppointmentItem object could be in the folder because of a previous response with **CdoResponseAccepted**, or because the recipient called GetAssociatedAppointment on the meeting request.

Calendar folders are not supported in the public folders store provided with Microsoft® Exchange, and AppointmentItem and MeetingItem objects are stored as Message objects. An attempt to access a property or method specific to an appointment or meeting, such as Duration or GetAssociatedAppointment, returns an error. If you have declared a MeetingItem object with early binding, for example `Dim objMeeting As MeetingItem`, CDO returns **CdoE_NO_SUPPORT** in this case. With late binding, Microsoft® Visual Basic® returns an error indicating no support.

A MeetingItem object can be rendered into HTML hypertext using the CDO Rendering ObjectRenderer object. To specify this, set the object renderer's DataSource property to the MeetingItem object itself. The individual properties that can be rendered with the RenderProperty method are indicated in the MeetingItem and Message object property descriptions.

A MeetingItem object can also be rendered as the parent of a Recipients collection, using the ContainerRenderer object. The individual properties that can be rendered with the RenderProperty method are indicated in the MeetingItem and Message object property descriptions.

GetAssociatedAppointment Method (MeetingItem Object) Group

The **GetAssociatedAppointment** method returns an AppointmentItem object associated with this meeting.

Syntax

Set *objAppointment* = *objMeeting*.**GetAssociatedAppointment**()

objAppointment

On successful return, contains the AppointmentItem object associated with this meeting.

objMeeting

Required. This MeetingItem object.

Remarks

When you receive a MeetingItem object representing a meeting request, you can call its **GetAssociatedAppointment** method to obtain an AppointmentItem object associated with the meeting request. This AppointmentItem is not the same object as the original AppointmentItem belonging to the meeting's **Organizer**, nor is it the same as an AppointmentItem obtained by any other recipient of the meeting request. Only the organizer can access the original AppointmentItem object.

The AppointmentItem as returned by **GetAssociatedAppointment** has the same property settings as the original, but you can change them as necessary. This does not affect any corresponding settings in the original AppointmentItem, or in any other recipient's associated AppointmentItem. Your AppointmentItem object is saved in your active calendar folder when you call its **Update** method.

If the meeting request is a meeting cancellation, that is, if its associated appointment's **MeetingStatus** property contains **CdoMeetingCanceled**, and you never received the original meeting request, **GetAssociatedAppointment** returns **CdoE_NOT_FOUND**. This behavior is compatible with Microsoft Schedule+ but not with Microsoft Outlook, which creates a tentative appointment in the calendar and brings up a form that asks the user to cancel this tentative appointment.

CDO automatically creates and saves an associated AppointmentItem for you if you call the MeetingItem object's **Respond** method with the *RespondType* parameter set to **CdoResponseAccepted** or **CdoResponseTentative**. Regardless of how the associated AppointmentItem is created, you are responsible for deleting it when you are finished with it, even if you subsequently decline the meeting request by calling **Respond** with *RespondType* set to **CdoResponseDeclined**.

If the meeting's organizer has sent an updated meeting request since you accepted the first request, CDO considers the existing AppointmentItem object in your calendar folder to be out of date. When you call **GetAssociatedAppointment**, the existing appointment is deleted from your calendar folder, and **GetAssociatedAppointment** returns a new, updated AppointmentItem object. However, this new appointment is not stored in your calendar folder until you call its **Update** method.

You can work directly with the AppointmentItem object to access its specifications and respond to it. The appointment returned by **GetAssociatedAppointment** is the only AppointmentItem object on which you can call the **Respond** method.

Another messaging user can delegate you to receive messages on behalf of that user. This delegation cannot be done with CDO but is possible using MAPI. If you have been so delegated and receive a meeting request for that user, you can view the MeetingItem object in your Inbox, but you cannot accept the meeting request. Similarly, if you access the Inbox of a messaging user that has delegated reception to you, you can view meeting requests but not respond to them. In either of these cases, an attempt to call **GetAssociatedAppointment** or **Respond** returns **CdoE_NO_SUPPORT**.

Calendar folders are not supported in the public folders store provided with Microsoft® Exchange, and MeetingItem objects are stored as Message objects. An attempt to call **GetAssociatedAppointment** in this case returns **CdoE_NO_SUPPORT**.

MeetingType Property (MeetingItem Object) Group

The **MeetingType** property returns the type of this meeting item. Read-only.

Syntax

objMeeting.MeetingType

Data Type

Long

Remarks

The **MeetingType** property is set automatically by the **Send** method when you send a meeting request from an [AppointmentItem](#) object and when you send a response from an AppointmentItem or MeetingItem object. **MeetingType** can have exactly one of the following values:

| MeetingType value | Decimal value | Description |
|---------------------------|---------------|---|
| CdoMeetingRequest | 1 | This meeting item is a meeting request or cancellation. |
| CdoMeetingResponse | 2 | This meeting item is a response to a meeting request. |

The message class of the MeetingItem object depends on whether it is being sent by the meeting organizer or by a recipient. Meeting items from the organizer have message class IPM.Schedule.Meeting.Request or IPM.Schedule.Meeting.Canceled, depending on the setting of the **MeetingStatus** property on the originating [AppointmentItem](#) object. Meeting items from a recipient have a message class as described in the [Respond](#) method.

Calendar folders are not supported in the public folders store provided with Microsoft® Exchange, and MeetingItem objects are stored as [Message](#) objects. An attempt to read **MeetingType** in this case returns **CdoE_NO_SUPPORT**.

Respond Method (MeetingItem Object)

Group

The **Respond** method returns a MeetingItem object for responding to this meeting request.

Syntax

Set *objMeetResp* = *objMeeting*.**Respond**(*RespondType*)

objMeetResp

Object. On successful return, contains a MeetingItem object that can be used to respond to the meeting request.

objMeeting

Required. This MeetingItem object.

RespondType

Required. Long. The value to send as the response.

Remarks

The **Respond** method prepares a meeting response which can be sent in answer to a meeting request using the **Forward** or **Send** method. The response takes the form of a MeetingItem object with the meeting's initiating user as a primary recipient. The initiating user is available through the **Organizer** property of the associated AppointmentItem object, which can be obtained from the **GetAssociatedAppointment** method.

The *RespondType* parameter can have exactly one of the following values:

| <i>RespondType</i> setting | Decimal value | Description |
|-----------------------------|---------------|---|
| CdoResponseAccepted | 3 | This messaging user wishes to firmly accept the meeting request. |
| CdoResponseDeclined | 4 | This messaging user wishes to decline the meeting request. |
| CdoResponseTentative | 2 | This messaging user wishes to tentatively accept the meeting request. |

The message class of the response you send depends on the value you specify in the *RespondType* parameter. It is IPM.Schedule.Meeting.Resp.Pos if you accept, IPM.Schedule.Meeting.Resp.Neg if you decline, or IPM.Schedule.Meeting.Resp.Tent if you accept tentatively.

You can respond to the same meeting request more than once. If you call the **Respond** method with the *RespondType* parameter set to **CdoResponseAccepted** or **CdoResponseTentative**, CDO creates an AppointmentItem object associated with the meeting and saves it in your active calendar folder. You can use the **Respond** method on this appointment to change your response.

If you **Respond** to a meeting request with **CdoResponseDeclined**, no AppointmentItem object is created, but any AppointmentItem already in the folder is left undeleted. Therefore, if you accept a request and subsequently decline it, or if you call **GetAssociatedAppointment** on the meeting request and then respond with **CdoResponseDeclined**, you must either **Delete** the associated AppointmentItem object yourself or leave it in the folder.

If you have declined a meeting request and subsequently wish to accept it, you cannot use any associated AppointmentItem object for your new response. However, if you have retained the

requesting MeetingItem object, you can use its **Respond** method to accept the request.

If a meeting request is for a recurring meeting, your response applies to the entire recurring series. If you respond with **CdoResponseAccepted** or **CdoResponseTentative**, you can subsequently select an individual recurrence and alter your response to **CdoResponseAccepted**, **CdoResponseTentative**, or **CdoResponseDeclined**.

You cannot call **Respond** on a meeting cancellation, that is, a meeting request with **CdoMeetingCanceled** in its associated appointment's **MeetingStatus** property.

Another messaging user can delegate you to receive messages on behalf of that user. This delegation cannot be done with CDO but is possible using MAPI. If you have been so delegated and receive a meeting request for that user, you can view the MeetingItem object in your Inbox, but you cannot accept the meeting request. Similarly, if you access the Inbox of a messaging user that has delegated reception to you, you can view meeting requests but not respond to them. In either of these cases, an attempt to call **GetAssociatedAppointment** or **Respond** returns **CdoE_NO_SUPPORT**.

Calendar folders are not supported in the public folders store provided with Microsoft® Exchange, and MeetingItem objects are stored as **Message** objects. An attempt to call **Respond** in this case returns **CdoE_NO_SUPPORT**.

Calling the **Respond** method is the same as calling **GetAssociatedAppointment** and then calling **Respond** on the AppointmentItem object.

Example

```
Dim objSess As Session
Dim objMtg As MeetingItem
Dim objAppt As AppointmentItem
Dim objResp As MeetingItem ' response to meeting request
On Error Resume Next

Set objSess = CreateObject("MAPI.Session")
objSess.Logon
Set objMtg = objSess.Inbox.Messages(1)

If objMtg Is Nothing Then
    MsgBox "No messages in Inbox"
    ' ... error exit ...
ElseIf objMtg.Class <> 27 Then ' CdoMeetingItem
    MsgBox "Message is not a meeting request or response"
    ' ... error exit ...
End If
MsgBox "Meeting is " & objMtg ' default property is .Subject

' Message exists and is a meeting; is it a request?
If objMtg.MeetingType <> 1 Then ' CdoMeetingRequest
    MsgBox "Meeting item is not a request"
    ' ... error exit ...
End If
Set objAppt = objMtg.GetAssociatedAppointment
MsgBox "Meeting times" & objAppt.StartTime & " - " & objAppt.EndTime _
    & "; recurring is " & objAppt.IsRecurring
' we can Respond from either the AppointmentItem or the MeetingItem
Set objResp = objMtg.Respond(3) ' CdoResponseAccepted
objResp.Text = "OK, I'll be there"
objResp.Send
```


Message Object

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

At a Glance

| | |
|----------------------------|---|
| Specified in type library: | CDO.DLL |
| First available in: | CDO Library version 1.0.a |
| Parent objects: | <u>Messages</u> collection |
| Child objects: | <u>Attachments</u> collection
<u>Fields</u> collection
<u>Recipients</u> collection |
| Default property: | <u>Subject</u> |

Properties

| Name | Available since version | Type | Access |
|---------------------------------|-------------------------|--|--|
| <u>Application</u> | 1.0.a | String | Read-only |
| <u>Attachments</u> | 1.0.a | Attachment object or Attachments collection object | Read-only |
| <u>Categories</u> | 1.2 | String array | Read/write |
| <u>Class</u> | 1.0.a | Long | Read-only |
| <u>Conversation</u> | 1.0.a | (Obsolete. Do not use.) | Read/write |
| <u>ConversationIndex</u> | 1.0.a | String | Read/write |
| <u>ConversationTopic</u> | 1.0.a | String | Read/write |
| <u>DeliveryReceipt</u> | 1.0.a | Boolean | Read/write |
| <u>Encrypted</u> | 1.0.a | Boolean | Read/write |
| <u>Fields</u> | 1.0.a | Field object or Fields collection object | Read-only |
| <u>FolderID</u> | 1.0.a | String | Read-only |
| <u>ID</u> | 1.0.a | String | Read-only |
| <u>Importance</u> | 1.0.a | Long | Read/write |
| <u>MAPIOBJECT</u> | 1.0.a | IUnknown object | Read/write
(Note: Not available to Visual Basic applications) |
| <u>Parent</u> | 1.0.a | Messages collection object | Read-only |
| <u>ReadReceipt</u> | 1.0.a | Boolean | Read/write |

| | | | |
|--------------------------------|-------|---|------------|
| <u>Recipients</u> | 1.0.a | Recipients object or Recipients collection object | Read/write |
| <u>Sender</u> | 1.0.a | AddressEntry object | Read/write |
| <u>Sensitivity</u> | 1.2 | Long | Read/write |
| <u>Sent</u> | 1.0.a | Boolean | Read/write |
| <u>Session</u> | 1.0.a | Session object | Read-only |
| <u>Signed</u> | 1.0.a | Boolean | Read/write |
| <u>Size</u> | 1.0.a | Long | Read-only |
| <u>StoreID</u> | 1.0.a | String | Read-only |
| <u>Subject</u> | 1.0.a | String | Read/write |
| <u>Submitted</u> | 1.0.a | Boolean | Read/write |
| <u>Text</u> | 1.0.a | String | Read/write |
| <u>TimeCreated</u> | 1.2 | Variant (vbDate format) | Read-only |
| <u>TimeExpired</u> | 1.2 | Variant (vbDate format) | Read/write |
| <u>TimeLastModified</u> | 1.2 | Variant (vbDate format) | Read-only |
| <u>TimeReceived</u> | 1.0.a | Variant (vbDate format) | Read/write |
| <u>TimeSent</u> | 1.0.a | Variant (vbDate format) | Read/write |
| <u>Type</u> | 1.0.a | String | Read/write |
| <u>Unread</u> | 1.0.a | Boolean | Read/write |

Methods

| Name | Available since version | Parameters |
|------------------------|--------------------------------|---|
| <u>CopyTo</u> | 1.1 | <i>folderID</i> as String ,
(optional) <i>storeID</i> as String |
| <u>Delete</u> | 1.0.a | (none) |
| <u>Forward</u> | 1.2 | (none) |
| <u>IsSameAs</u> | 1.1 | <i>objMessage2</i> as Object |
| <u>MoveTo</u> | 1.1 | <i>folderID</i> as String ,
(optional) <i>storeID</i> as String |
| <u>Options</u> | 1.0.a | (optional) <i>parentWindow</i> as Long |
| <u>Reply</u> | 1.2 | (none) |
| <u>ReplyAll</u> | 1.2 | (none) |
| <u>Send</u> | 1.0.a | (optional) <i>saveCopy</i> as Boolean ,
(optional) <i>showDialog</i> as Boolean ,
(optional) <i>parentWindow</i> as Long |
| <u>Update</u> | 1.0.a | (optional) <i>makePermanent</i> as |

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

Remarks

Microsoft® Visual Basic® programmers can create new Message objects using the Messages collection's **Add** method.

C/C++ programmers can create new Message objects with the OLE function **CoCreateInstance**.

A message can be obtained from its parent Messages collection using the collection's Item property. To get to the Messages collection in a folder, use the Folder object's **Messages** property. If you know a message's unique identifier, you can obtain it directly from the Session object's **GetMessage** method.

A Message object can be rendered into HTML hypertext using the CDO Rendering ObjectRenderer object. To specify this, set the object renderer's **DataSource** property to the Message object itself. The individual properties that can be rendered with the **RenderProperty** method are indicated in the Message object property descriptions.

A Message object can also be rendered as the parent of a Recipients collection, using the ContainerRenderer object. The individual properties that can be rendered with the **RenderProperty** method are indicated in the Message object property descriptions.

Attachments Property (Message Object)

Group

The **Attachments** property returns a single [Attachment](#) object or an [Attachments](#) 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.

Data Type

Object (Attachment or Attachments collection)

Remarks

You can change individual Attachment objects within the Attachments collection, **Add** them to the collection, and **Delete** them from the collection. However, unlike the **Recipients** property, the **Attachments** property is read-only, so you cannot, for example, copy the complete Attachments collection to another message with a single instruction.

Although the **Attachments** property itself is read-only, the collection it returns can be accessed in the normal manner through its **Add** and **Delete** methods, and the properties on its member [Attachment](#) objects retain their respective read/write or read-only accessibility.

The **Attachments** property does not correspond to a MAPI property and cannot be rendered into HTML hypertext by the CDO Rendering Library. If a single [Attachment](#) object is returned, it could be rendered as an object by setting the [ObjectRenderer](#) object's **DataSource** property to the Attachment object returned by the **Attachments** property.

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)
```


Categories Property (Message Object)

Group

The **Categories** property specifies the categories assigned to the message. Read/write.

Syntax

objMessage.Categories

Data Type

String array

Remarks

The contents of the **Categories** property are defined by the application. **Categories** is commonly used to hold a set of keywords that can be used to access messages in a folder.

AppointmentItem objects in a Microsoft® Schedule+ calendar folder do not have the full set of attributes of a general message. If you obtain the default calendar folder by passing **CdoDefaultFolderCalendar** to the Session object's **GetDefaultFolder** method, its appointments have no defined value for the **Categories** property. An attempt to access **Categories** in this case returns **CdoE_NO_SUPPORT**.

When you declare a variable to interact with the **Categories** property, you must **Dim** it appropriately, not just as an array of unspecified data type. To copy the variable to **Categories**, declare it as a **String** array:

```
Dim KWords (10) As String ' NOT just Dim KWords (10)
' ...
objMessage.Categories = KWords ' accepts string array
```

To copy **Categories** to a variable, declare the variable as a **Variant** array:

```
Dim Keys (10) As Variant ' NOT Dim Keys (10) or Dim As String
' ...
Keys = objMessage.Categories ' returns variant array
```

This is because **Categories** accepts arrays of various data types but always returns a **Variant** array.

Note When you access an element of the **Categories** array, whether to read it or write it, you must include an extra set of parentheses to satisfy the Visual Basic compiler's allowance for a property parameter:

```
objMessage.Categories()(3) = "Third category"
' ...
Kwords(i) = objMessage.Categories()(i)
```

For more information on property parameters, see [Property Parameters](#).

Example

This code fragment sets the **Categories** property of a message from a string array of keywords and later displays the categories:

```
Dim objMessage As Message
' assume objMessage is valid and already being accessed
Dim KWords (10) As Variant ' NOT As String or just Dim KWords(10)
' ... obtain up to ten keywords from user ...
objMessage.Categories = KWords ' whole array; no index or parameter
```

```
' ... later, read the message and examine its properties ...  
MsgBox "Categories for this message are:"  
For i = LBound(objMessage.Categories) To UBound(objMessage.Categories)  
    If 0 < Len(objMessage.Categories()(i)) Then  
        MsgBox vbCrLf & objMessage.Categories()(i)  
    End If  
Next i
```

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) Group

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 responses in their personal folders. Messages can be tagged with the **ConversationIndex** property so that users can order the messages within the conversation.

The **Session** object provides the **CreateConversationIndex** method to create or update a conversation index.

This convention uses concatenated time stamp values, with each new message in the conversation adding a new time stamp to the end of the **ConversationIndex** string. You can see time relationships among the messages when you sort them by **ConversationIndex** values.

For more information on conversations, see [Working With Conversations](#).

The **ConversationIndex** property is not exposed on **AppointmentItem** objects created by Microsoft® Schedule+, and it is not automatically set when you create an appointment within a CDO application. In these cases, the application must assign a value to **ConversationIndex** or an attempt to read it returns **CdoE_NOT_FOUND**.

The **ConversationIndex** property corresponds to the MAPI property PR_CONVERSATION_INDEX. It can be rendered into HTML hypertext using the CDO Rendering **ObjectRenderer** object. To specify this, set the object renderer's **DataSource** property to this Message object and the *property* parameter of the **RenderProperty** method to **CdoPR_CONVERSATION_INDEX**.

Example

This code fragment demonstrates the old procedure used prior to version 1.1 of CDO. It takes advantage of the OLE **CoCreateGUID** function, which returns a value that consists of a time stamp and a machine identifier. The code fragment saves the time stamp part of the **GUID**.

For an example of the new procedure available with CDO version 1.1, see the **CreateConversationIndex** method.

```
' 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 IResult As Long
Dim IGuid As GUID
' Exchange conversation is a unique 8-byte value
' Exchange client viewer sorts by concatenated properties
On Error GoTo error_actmsg

IResult = CoCreateGuid(IGuid)
If IResult = S_OK Then
    Util_GetEightByteTimeStamp = _
        Hex$(IGuid.Guid1) & Hex$(IGuid.Guid2)
Else
    Util_GetEightByteTimeStamp = "00000000" ' zero time stamp
End If
Exit Function
```

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

```
Function Util_NewConversation()
Dim i As Integer
Dim objNewMsg As Message ' new message object
Dim strNewIndex As String ' value for ConversationIndex
' ... error handling ...
Set objNewMsg = objSession.Outbox.Messages.Add
' ... error handling ...
With objNewMsg
    .Subject = "used car wanted"
    .ConversationTopic = .Subject
    .ConversationIndex = Util_GetEightByteTimeStamp() ' utility
    .Text = "Wanted: late-model used car with low mileage."
    Set objOneRecip = .Recipients.Add(Name:="Car Ads", Type:=CdoTo)
    ' 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 Folder
Dim i As Integer
```

```

Dim objOriginalMsg As Message ' original message in public folder
Dim objNewMsg As Message      ' 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 bicycle?" ' 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:=CdoTo)
' ... 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) Group

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 considered 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 a value appropriate to all messages within the conversation, not only to the first message. For many applications, the message's **Subject** property is appropriate.

CDO does not automatically copy the **ConversationTopic** property to other messages unless you are making an exact copy with the **CopyTo** method. When your application creates 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 is not exposed on [AppointmentItem](#) objects created by Microsoft® Schedule+, and it is not automatically set when an application creates an appointment in a Schedule+ calendar folder. In these cases, the application must assign a value to **ConversationTopic** or an attempt to read it returns **CdoE_NOT_FOUND**. If you create an appointment in a Microsoft® Outlook™ calendar folder, **ConversationTopic** is initially set to an empty string.

The **ConversationTopic** property corresponds to the MAPI property PR_CONVERSATION_TOPIC. It can be rendered into HTML hypertext using the CDO Rendering [ObjectRenderer](#) object. To specify this, set the object renderer's **DataSource** property to this Message object and the *property* parameter of the [RenderProperty](#) method to **CdoPR_CONVERSATION_TOPIC**.

Example

See the examples for the **ConversationIndex** property and for the [Session](#) object's [CreateConversationIndex](#) method.

CopyTo Method (Message Object)

Group

The **CopyTo** method makes a copy of the Message object in another folder.

Syntax

Set *objCopiedMessage* = *objMessage*.**CopyTo**(*folderID* [, *storeID*])

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

The current version of CDO does not support the **CopyTo** method on AppointmentItem objects.

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)

Group

The **Delete** method removes the [AppointmentItem](#), [MeetingItem](#), or Message object from the [Messages](#) collection.

Syntax

```
objMessage.Delete( )
```

Remarks

The **Delete** method performs an irreversible operation on the collection. It calls **Release** on the collection's reference to the [AppointmentItem](#), [MeetingItem](#), or Message object. If you have another reference to the object, you can still access its properties and methods, but you can never again associate it with any collection because the **Add** method always creates a new object. You should **Set** your reference variable either to **Nothing** or to another object.

The final **Release** on the [AppointmentItem](#), [MeetingItem](#), or Message object takes place when you assign your reference variable to **Nothing**, or when you call **Delete** if you had no other reference. At this point the object is removed from memory. Attempted access to a released object results in an error return of **CdoE_INVALID_OBJECT**.

The action of the **Delete** method is permanent, and the Message object cannot be restored to the collection. Before calling **Delete**, your application can prompt the user to verify whether the message should be permanently deleted.

The **Delete** method does not move a [MeetingItem](#) or Message object to the Deleted Items folder. If this folder exists and you wish this functionality, you must specifically program your application for it.

If the Messages collection underlies a categorized view and the deleted message was the last remaining message in a group, the group's [GroupHeader](#) object is also removed from the collection. Unlike the message, however, it is not moved anywhere. Group headers do not persist in storage, and when the collection is released, whether with messages or not, the [GroupHeader](#) objects cease to exist.

When you delete a member of a collection, the collection is immediately refreshed, meaning that its **Count** property is reduced by one and its members are reindexed. To access a member following the deleted member, you must use its new index value. For more information, see [Looping Through a Collection](#).

You can delete all the group headers and 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.

Example

This code fragment illustrates the two situations previously explained. The **Set** statement calls **AddRef** on the first Message object. That reference survives the call to **Delete** and has to be reassigned. The second Message object is deleted without creating another reference, and no other action is necessary.

```
' assume valid Folder object
Set objMessage = objFolder.Messages.Item(1)
objMessage.Delete ' still have a reference from Set statement
' ... other operations on objMessage possible but pointless ...
Set objMessage = Nothing ' necessary to remove reference
' ...
objFolder.Messages.Item(2).Delete ' no reference to remove
```

DeliveryReceipt Property (Message Object) Group

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 each recipient receives your message. The default setting is **False**.

Every transport provider that handles your message sends you a single delivery notification containing the names and addresses of all the recipients that provider delivered your message to. Therefore you might not get a separate notification for each recipient. 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](#).

Not all transport providers support notification requests.

The **DeliveryReceipt** property corresponds to the MAPI property PR_ORIGINATOR_DELIVERY_REPORT_REQUESTED. It can be rendered into HTML hypertext using the CDO Rendering [ObjectRenderer](#) object. To specify this, set the object renderer's **DataSource** property to this Message object and the *property* parameter of the [RenderProperty](#) method to **CdoPR_ORIGINATOR_DELIVERY_REPORT_REQUESTED**.

Encrypted Property (Message Object)

Group

The **Encrypted** property is **True** if the message has been encrypted or if encryption is being requested. Read/write.

Syntax

objMessage.Encrypted

Data Type

Boolean

Remarks

The **Encrypted** property represents only a request for encryption. The effect of setting it is dependent upon the message store or transport provider. CDO does not encrypt or digitally sign the message.

Security features include the **Encrypted** and **Signed** properties.

The **Encrypted** property corresponds to the SECURITY_ENCRYPTED flag of the MAPI property PR_SECURITY. It can be rendered into HTML hypertext using the CDO Rendering ObjectRenderer object. To specify this, set the object renderer's **DataSource** property to this Message object and the *property* parameter of the **RenderProperty** method to **CdoPR_SECURITY**. However, you must assign a CDO Rendering Format object to the PR_SECURITY property and use the **Value** property of the format's Pattern objects to isolate the SECURITY_ENCRYPTED flag's setting for rendering.

Fields Property (Message Object)

Group

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

Integer. Value must be greater than 0 and less than or equal to 65,535 (&HFFFF). Specifies the index within the Fields collection.

proptag

Long. Value must be greater than or equal to 65,536 (&H10000). 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 (Field or Fields collection)

Remarks

The **Fields** property returns one or all of the fields associated with a Message object. Each field typically corresponds to a MAPI property. Data types are preserved, except that MAPI counted binary properties are converted to and from character strings representing hexadecimal digits.

The **Fields** property provides a generic access mechanism that allows Microsoft® Visual Basic® and Microsoft® 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 **CdoPR_PRIORITY**. To access a named property, use *objMessage.Fields(name)*, where *name* is a string that represents the custom property name.

Although the **Fields** property itself is read-only, the collection it returns can be accessed in the normal manner through its **Add** and **Delete** methods, and the properties on its member [Field](#) objects retain their respective read/write or read-only accessibility.

The **Fields** property does not correspond to a MAPI property and cannot be rendered into HTML hypertext by the CDO Rendering Library.

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)
' could also be objFieldsColl(i) since Item is default property
If objOneField Is Nothing Then
    MsgBox "error; cannot get this Field object"
Else
    MsgBox "Selected Field " & i
End If
```


FolderID Property (Message Object)

Group

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. However, MAPI does not require identifier values to be binary comparable. Accordingly, two identifier values can be different, yet refer to the same object. MAPI compares identifiers with the **CompareEntryIDs** method. CDO provides the **CompareIDs** method in the [Session](#) object. For more information on entry identifiers, see the *MAPI Programmer's Reference*.

The **FolderID** property is not exposed on [AppointmentItem](#) objects created by Microsoft® Schedule+, and it is not automatically set when you create an appointment within a CDO application until you call the [Update](#) method. In these cases, the application must assign a value to **FolderID** or an attempt to read it returns **CdoE_NOT_FOUND**.

The **FolderID** property corresponds to the MAPI property PR_PARENT_ENTRYID, converted to a string of hexadecimal characters. It can be rendered into HTML hypertext using the CDO Rendering [ObjectRenderer](#) object. To specify this, set the object renderer's **DataSource** property to this Message object and the *property* parameter of the [RenderProperty](#) method to **CdoPR_PARENT_ENTRYID**.

Forward Method (Message Object)

Group

The **Forward** method returns a new Message object that can be used to forward the current message.

Syntax

Set *objForwardMessage* = *objMessage*.**Forward**()

objForwardMessage

On successful return, contains the new Message object ready for forwarding.

objMessage

Required. This Message object.

Remarks

The **Forward** method copies the current message to the new message, including any Attachment objects associated with it. In keeping with forwarding conventions, **Forward** does not copy any Recipient objects to the new message. You must populate the Recipients collection before calling the **Send** method.

Forward does not save any changes or send any e-mail. You must call **Update** on the new Message object to save it in the messaging system, or **Send** to initiate transmission to the recipients.

The current implementation of the **Forward** method does not copy the Text property to the new message. If you call the **Forward** method on a MeetingItem object, it returns another MeetingItem object instead of a Message object. A recurring meeting is forwarded, but any instantiated individual recurrences are not.

If you call **Forward** on an AppointmentItem object, it returns **CdoE_NO_SUPPORT** if the appointment's MeetingStatus property is set to **CdoNonMeeting**. Otherwise it returns a MeetingItem object. This allows you to forward a meeting request that you have accepted.

Microsoft® Outlook™ deletes an accepted meeting request but leaves the resulting AppointmentItem intact with its MeetingStatus set to **CdoMeeting**. However, if you call **GetAssociatedAppointment** on the MeetingItem object returned by **Forward**, the AppointmentItem object is deleted. This is because CDO considers the new AppointmentItem object returned by **GetAssociatedAppointment** to be more recent. You can prevent this behavior by calling **Update** on the AppointmentItem object you wish to retain before you call **GetAssociatedAppointment** on the MeetingItem object.

ID Property (Message Object)

Group

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. However, MAPI does not require identifier values to be binary comparable. Accordingly, two identifier values can be different, yet refer to the same object. MAPI compares identifiers with the **CompareEntryIDs** method. CDO provides the [CompareIDs](#) method in the [Session](#) object. For more information on entry identifiers, see the *MAPI Programmer's Reference*.

The **ID** property is not automatically set when you create an appointment within a CDO application until you call the [Update](#) method. In this case, the application must assign a value to **ID** or an attempt to read it returns **CdoE_NOT_FOUND**.

The **ID** property corresponds to the MAPI property PR_ENTRYID, converted to a string of hexadecimal characters. It can be rendered into HTML hypertext using the CDO Rendering [ObjectRenderer](#) object. To specify this, set the object renderer's [DataSource](#) property to this Message object and the *property* parameter of the [RenderProperty](#) method to **CdoPR_ENTRYID**.

Example

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

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


Importance Property (Message Object)

Group

The **Importance** property returns or sets the importance of the message. Read/write.

Syntax

objMessage.Importance

Data Type

Long

Remarks

The **Importance** property can have exactly one of the following values:

| Constant | Value | Description |
|------------------|-------|-----------------------------|
| CdoLow | 0 | Low importance |
| CdoNormal | 1 | Normal importance (default) |
| CdoHigh | 2 | High importance |

AppointmentItem objects in a Microsoft® Schedule+ calendar folder do not have the full set of attributes of a general message. If you obtain the default calendar folder by passing **CdoDefaultFolderCalendar** to the Session object's **GetDefaultFolder** method, its appointments have no defined value for the **Importance** property. An attempt to access **Importance** in this case returns **CdoE_NOT_FOUND**.

The **Importance** property corresponds to the MAPI property PR_IMPORTANCE. It can be rendered into HTML hypertext using the CDO Rendering ObjectRenderer object. To specify this, set the object renderer's **DataSource** property to this Message object and the *property* parameter of the **RenderProperty** method to **CdoPR_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 = "Meeting change"  
  objMessage.Text = "Reschedule to Wednesday at noon."  
  objMessage.Importance = CdoHigh  
  objMessage.Send
```

See Also

Send Method (Message Object)

IsSameAs Method (Message Object)

Group

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

Syntax

```
boolSame = objMessage.IsSameAs(objMessage2)
```

boolSame

On successful return, contains **True** if the two objects are the same.

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 in the underlying messaging system. 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. This is necessary because, although MAPI requires all entry identifiers to be unique, it does not require two of them identifying the same object to be identical. A generic comparison of any two objects' unique identifiers is also available with the Session object's **CompareIDs** method.

MAPIOBJECT Property (Message Object)

Group

The **MAPIOBJECT** property returns an **IUnknown** pointer to the Message object. Not available to Microsoft® 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 accessible only by C/C++ programs that deal with **IUnknown** objects. Visual Basic supports the **IDispatch** interface and not **IUnknown**. The **MAPIOBJECT** property is an **IUnknown** object that returns an **IMessage** interface in response to **QueryInterface**. For more information, see [Introduction to Automation](#) and [How Programmable Objects Work](#). Also see the "COM and ActiveX Object Services" section of the Microsoft Platform SDK.

If your application uses any **MAPIOBJECT** or **RawTable** properties, it must **Release** them all before calling the [Session](#) object's **Logoff** method. Failure to do so can result in unexpected behavior.

The **MAPIOBJECT** property does not correspond to a MAPI property and cannot be rendered into HTML hypertext by the CDO Rendering Library.

MoveTo Method (Message Object)

Group

The **MoveTo** method relocates the Message object to another folder.

Syntax

Set *objMovedMessage* = *objMessage*.**MoveTo**(*folderID* [, *storeID*])

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

The current version of CDO does not support the **MoveTo** method on AppointmentItem objects.

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)

Group

The **Options** method displays a modal dialog box where the user can change the submission options for a message.

Syntax

objMessage.Options([*parentWindow*])

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 box should be application-modal.

Remarks

The **Options** dialog box is always modal, meaning the parent window is disabled while the dialog box 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 box is active. If the *parentWindow* parameter is supplied but is not valid, the call returns **CdoE_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 **CdoE_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 **Options** method cannot be called on an [AppointmentItem](#) or [MeetingItem](#) object. Any attempted call from such an object returns **CdoE_NO_SUPPORT**.

The following methods can invoke dialog boxes:

- **Details** method ([AddressEntry](#) object)
- **Options** and **Send** methods (Message object)
- **Resolve** method ([Recipient](#) object)
- **Resolve** method ([Recipients](#) collection)
- **AddressBook** and **Logon** methods ([Session](#) object)

However, if your application is running as a Microsoft® Windows NT® service, for example from Active Server Pages (ASP) script on a Microsoft® Internet Information Server (IIS), no user interface is allowed.

For more information on Windows NT services, see the Win32® Web page *Using MAPI from a Windows NT Service* at <http://www.microsoft.com/win32dev/mapi/mapiserv.htm>. For more information on running as a service, see "Windows NT Service Client Applications" in the *MAPI Programmer's Reference*.

ReadReceipt Property (Message Object)

Group

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 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](#).

Not all transport providers support notification requests.

The **ReadReceipt** property corresponds to the MAPI property PR_READ_RECEIPT_REQUESTED. It can be rendered into HTML hypertext using the CDO Rendering [ObjectRenderer](#) object. To specify this, set the object renderer's **DataSource** property to this Message object and the *property* parameter of the [RenderProperty](#) method to **CdoPR_READ_RECEIPT_REQUESTED**.

Recipients Property (Message Object)

Group

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 (Recipient or Recipients collection)

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 Microsoft® Visual Basic® instruction. For example, you can copy the complete recipient list of a received message, with all of each recipient's properties, to a reply message:

```
Set objReplyMsg.Recipients = objReceivedMsg.Recipients
Set objSenderAE = objReceivedMsg.Sender
Set objOrigSender = objReplyMsg.Recipients.Add(objSenderAE.ID)
' then copy important properties from objSenderAE
```

Note that, unlike the **Recipients** property, the **Attachments** property is read-only, so the Attachments collection cannot be copied as a whole. You must deal with attachments in the manner of the following example.

The **Recipients** property does not correspond to a MAPI property and cannot be rendered into HTML hypertext by the CDO Rendering Library. If a [Recipients](#) collection is returned, it could be rendered as a container object by setting the [ContainerRenderer](#) object's **DataSource** property to the Recipients collection object returned by the **Recipients** property.

Example

This code fragment uses a loop to create a copy of every valid recipient of the original message *objOneMsg* in the copy message *objCopyMsg*. For each copied recipient, it also copies important properties from the original. Note how much more code this requires than copying the **Recipients** property from the original message.

```
' from the sample function Util_CopyMessage
For i = 1 To objOneMsg.Recipients.Count Step 1
    Set objOneRecip = objOneMsg.Recipients.Item(i)
    If objOneRecip Is Not Nothing Then
        Set objCopyRecip = objCopyMsg.Recipients.Add
```

```
If objCopyRecip Is Nothing Then
    MsgBox "Unable to create recipient in message copy"
    Exit Function
End If
' Now copy the most important properties
objCopyRecip.Address = objOneRecip.Address
objCopyRecip.Name = objOneRecip.Name
objCopyRecip.Type = objOneRecip.Type
End If
Next i
```


Reply Method (Message Object)

Group

The **Reply** method returns a new Message object that can be used to reply to the sender of the current message.

Syntax

Set *objReplyMessage* = *objMessage*.Reply()

objReplyMessage

On successful return, contains the new Message object ready for replying.

objMessage

Required. This Message object.

Remarks

The **Reply** method copies the current message to the new message and populates its Recipients collection with a single "To" recipient set from the original message's **Sender** property. In keeping with response conventions, **Reply** does not copy any Attachment objects to the new message.

Reply does not save any changes or send any e-mail. You must call Update on the new Message object to save it in the messaging system, or Send to initiate transmission to the recipients.

The current implementation of the **Reply** method does not copy the Text property to the new message.

ReplyAll Method (Message Object)

Group

The **ReplyAll** method returns a new Message object that that can be used to reply to the sender and all recipients of the current message.

Syntax

Set *objReplyMessage* = *objMessage*.ReplyAll()

objReplyMessage

On successful return, contains the new Message object ready for replying.

objMessage

Required. This Message object.

Remarks

The **ReplyAll** method copies the current message to the new message and populates its Recipients collection appropriately from the original message's Recipients and Sender properties. In keeping with response conventions, **ReplyAll** does not copy any Attachment objects to the new message.

ReplyAll does not save any changes or send any e-mail. You must call Update on the new Message object to save it in the messaging system, or Send to initiate transmission to the recipients.

The current implementation of the **ReplyAll** method does not copy the Text property to the new message.

Send Method (Message Object)

Group

The **Send** method sends the message to the recipients through the MAPI system.

Syntax

```
objMessage.Send( [saveCopy] [, showDialog] [, parentWindow] )
```

objMessage

Required. The Message object.

saveCopy

Optional. Boolean. If **True**, saves a copy of the message in a user folder, such as the Sent Items folder. The default value is **True**.

showDialog

Optional. Boolean. If **True**, displays a **Send Message** dialog box where the user can change the message contents or recipients. *showDialog* cannot be set to **True** when sending an [AppointmentItem](#) or [MeetingItem](#) object. 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 box should be application-modal. The *parentWindow* parameter is ignored unless *showDialog* is **True**.

Remarks

Like the **Update** method, the **Send** method saves all changes to the message in the MAPI system, but **Send** also moves the message to the current user's Outbox folder. Messaging systems retrieve messages from the Outbox and transport them to the recipients. After it is transported, a message is removed from the Outbox and deleted unless *saveCopy* is **True**.

You should compose your new messages in either your Inbox or your Outbox. The **Send** method normally deals only with messages located in these folders. However, if you have appointments in a calendar folder, you can send them from that folder.

If you are sending a meeting request from an [AppointmentItem](#) object, you must first set the appointment's **MeetingStatus** property to **CdoMeeting**, or the call to **Send** returns **CdoE_NO_SUPPORT**. Also, there is no form for showing a meeting request, so you must either set the *showDialog* parameter to **False** or let it default. If *showDialog* is **True**, **Send** returns **CdoE_NO_SUPPORT**.

While you are still composing a message you can set its **Sent** property to **False** or leave it unset. The **Send** method sets the **Submitted** property to **True** when the message is accepted in the message store. After the message is transported, the spooler function of the sending messaging system sets the **Sent** and **Unread** properties to **True**. The receiving messaging system sets the **Submitted** property to **False**.

The **Send** method invalidates the composed 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 sent message results in a return of **CdoE_INVALID_OBJECT**.

The **Send** dialog box is always modal, meaning the parent window is disabled while the dialog box 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 box is active. If the *parentWindow* parameter is supplied but is not valid, the call returns **CdoE_INVALID_PARAMETER**.

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 dialog boxes:

- **Details** method (AddressEntry object)
- **Options** and **Send** methods (Message object)
- **Resolve** method (Recipient object)
- **Resolve** method (Recipients collection)
- **AddressBook** and **Logon** methods (Session object)

However, if your application is running as a Microsoft® Windows NT® service, for example from Active Server Pages (ASP) script on a Microsoft® Internet Information Server (IIS), no user interface is allowed.

For more information on Windows NT services, see the Win32® Web page *Using MAPI from a Windows NT Service* at <http://www.microsoft.com/win32dev/mapi/mapiserv.htm>. For more information on running as a service, see "Windows NT Service Client Applications" in the *MAPI Programmer's Reference*.

Sender Property (Message Object)

Group

The **Sender** property returns or sets the sender of a message as an [AddressEntry](#) object. Read/write.

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 ([AddressEntry](#))

Remarks

You can change the **Sender** property before a message is either sent or saved, for example in a public folder. After a message has been sent or saved, any attempt to change its **Sender** property is ignored.

You can use the **Sender** property to send or post a message from one messaging user on behalf of another messaging user. When you set **Sender** to a messaging user, CDO identifies that user as both the sender of the message and the user on whose behalf the message is being sent or posted. If you then set **Sender** to a second messaging user, the message is identified as being sent or posted by the second messaging user on behalf of the first:

```
Dim objMsg As MAPI.Message ' message to be sent/posted; assume valid
Dim objMyself As MAPI.AddressEntry ' I will send/post the message ...
Dim objMyBoss As MAPI.AddressEntry ' ... on behalf of my boss
' ...
Set objMyself = objSession.CurrentUser ' AE object for myself
Set objMyBoss = objMyself.Manager ' my boss; assume valid
With objMsg
    Set Sender = objMyBoss ' sets represented sender properties
    Set Sender = objMyself ' leaves represented sender properties alone
    .Subject = "I am sending/posting this on behalf of my manager"
End With
```

For more information, see "Sender Properties" and "Represented Sender Properties" in the *MAPI Programmer's Reference*.

The **Sender** property is not exposed on [AppointmentItem](#) objects created by Microsoft® Schedule+, and it is not automatically set when you create an appointment within a CDO application. In these cases, the application must assign a value to **Sender** or an attempt to read it returns **CdoE_NOT_FOUND**.

The **Sender** property does not correspond to a MAPI property and cannot be rendered into HTML hypertext by the CDO Rendering Library. It could be rendered as an object by setting the [ObjectRenderer](#) object's [DataSource](#) property to the [AddressEntry](#) object returned by the **Sender** property.

The [Name](#) property of the [AddressEntry](#) object returned by the **Sender** property corresponds to the MAPI property PR_SENT_REPRESENTING_NAME. To render just the sender's name, you can set the object renderer's [DataSource](#) property to this [AddressEntry](#) object and the *property* parameter of the

RenderProperty method to **CdoPR_SENT_REPRESENTING_NAME**.

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 AddressEntry object from the Sender"
    Exit Function
End If
MsgBox "Message was sent by " & objAddrEntry.Name
```

Sensitivity Property (Message Object)

Group

The **Sensitivity** property specifies the sensitivity of the message. Read/write.

Syntax

objMessage.Sensitivity

Data Type

Long

Remarks

The **Sensitivity** property can have exactly one of the following values:

| Constant | Value | Description |
|-------------------------|-------|------------------------------------|
| CdoNoSensitivity | 0 | No special sensitivity (default) |
| CdoPersonal | 1 | Personal |
| CdoPrivate | 2 | Private |
| CdoConfidential | 3 | Designated as company confidential |

AppointmentItem objects in a Microsoft® Schedule+ calendar folder do not have the full set of attributes of a general message. If you obtain the default calendar folder by passing **CdoDefaultFolderCalendar** to the Session object's **GetDefaultFolder** method, its appointments have no defined value for the **Sensitivity** property. If you set **Sensitivity** to **CdoPersonal** or **CdoConfidential**, Schedule+ stores it as **CdoNoSensitivity**.

The **Sensitivity** property corresponds to the MAPI property PR_SENSITIVITY. It can be rendered into HTML hypertext using the CDO Rendering ObjectRenderer object. To specify this, set the object renderer's **DataSource** property to this Message object and the *property* parameter of the **RenderProperty** method to **CdoPR_SENSITIVITY**.

Example

This code fragment sets the sensitivity of a message as personal:

```
' from the sample function QuickStart:  
Set objMessage = objSession.Outbox.Messages.Add  
' ... check here to verify the message was created ...  
objMessage.Subject = "Meeting change"  
objMessage.Text = "Reschedule to Wednesday at noon."  
objMessage.Sensitivity = CdoPersonal  
objMessage.Send
```

Sent Property (Message Object)

Group

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 messages, you use the **Sent**, **Submitted**, and **Unread** properties and the **Send** or **Update** methods.

The following table summarizes the use of the message properties and methods for the three kinds of messages. In some systems the message store and transport providers are tightly coupled, in which case they bypass the spooler and perform its functions themselves.

| Kind of message | Method used | Sent property | Submitted property | Unread property |
|-----------------|---------------|-------------------------------|-------------------------------|------------------------------|
| Sent | Send | Spooler sets True | Send sets True | Spooler sets True |
| Posted | Update | Application sets True | Application sets False | 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 you call the **Send** 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 **Send**, the messaging system controls the **Sent** property and changes it to a read-only property. The receiving messaging system resets the **Submitted** property to **False** when the message arrives in a recipient's Inbox.

A common use for writing a value to the **Sent** property is to set it 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. You can cause unexpected results if you set the property incorrectly.

For posted messages, you create the message directly within a public folder and call the **Update** method. Some viewers do not allow the message to become visible to other users until you set the **Submitted** property to **True**.

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 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 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 is not exposed on AppointmentItem objects created by Microsoft® Schedule+, and it is not automatically set when you create an appointment within a CDO application until you call the Update method. In these cases, the application must assign a value to **Sent** or an attempt to read it returns **CdoE_NOT_FOUND**.

The **Sent** property corresponds to the MSGFLAG_UNSENT flag not being set in the MAPI property PR_MESSAGE_FLAGS. It can be rendered into HTML hypertext using the CDO Rendering ObjectRenderer object. To specify this, set the object renderer's **DataSource** property to this Message object and the *property* parameter of the RenderProperty method to **CdoPR_MESSAGE_FLAGS**. However, you must assign a CDO Rendering Format object to the PR_MESSAGE_FLAGS property and use the **Value** property of the format's Pattern objects to isolate the MSGFLAG_UNSENT flag's setting for rendering.

Signed Property (Message Object)

Group

The **Signed** property is **True** if the message has been tagged with a digital signature or if digital signing is being requested. Read/write.

Syntax

objMessage.Signed

Data Type

Boolean

Remarks

The **Signed** property represents only a request for a signature. The effect of setting it is dependent upon the message store or transport provider. CDO does not encrypt or digitally sign the message.

Security features include the **Encrypted** and **Signed** properties.

The **Signed** property corresponds to the SECURITY_SIGNED flag of the MAPI property PR_SECURITY. It can be rendered into HTML hypertext using the CDO Rendering ObjectRenderer object. To specify this, set the object renderer's **DataSource** property to this Message object and the *property* parameter of the **RenderProperty** method to **CdoPR_SECURITY**. However, you must assign a CDO Rendering Format object to the PR_SECURITY property and use the **Value** property of the format's Pattern objects to isolate the SECURITY_SIGNED flag's setting for rendering.

Size Property (Message Object)

Group

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 is not exposed on **AppointmentItem** objects created by Microsoft® Schedule+, and it is not automatically set when you create an appointment within a CDO application until you call the **Update** method. In these cases, the application must assign a value to **Size** or an attempt to read it returns **CdoE_NOT_FOUND**.

The **Size** property corresponds to the MAPI property PR_MESSAGE_SIZE. It can be rendered into HTML hypertext using the CDO Rendering **ObjectRenderer** object. To specify this, set the object renderer's **DataSource** property to this Message object and the *property* parameter of the **RenderProperty** method to **CdoPR_MESSAGE_SIZE**.

StoreID Property (Message Object)

Group

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 is not exposed on AppointmentItem objects created by Microsoft® Schedule+, and it is not automatically set when you create an appointment within a CDO application until you call the **Update** method. In these cases, the application must assign a value to **StoreID** or an attempt to read it returns **CdoE_NOT_FOUND**.

The **StoreID** property corresponds to the MAPI property PR_STORE_ENTRYID, converted to a string of hexadecimal characters. It can be rendered into HTML hypertext using the CDO Rendering ObjectRenderer object. To specify this, set the object renderer's **DataSource** property to this Message object and the *property* parameter of the RenderProperty method to **CdoPR_STORE_ENTRYID**.

Subject Property (Message Object)

Group

The **Subject** property returns or sets the subject of the message as a string. Read/write.

Syntax

objMessage.Subject

The **Subject** property is the default property of a Message object, meaning that *objMessage* is syntactically equivalent to *objMessage.Subject* in Microsoft® Visual Basic® code.

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. It can be rendered into HTML hypertext using the CDO Rendering ObjectRenderer object. To specify this, set the object renderer's **DataSource** property to this Message object and the *property* parameter of the **RenderProperty** method to **CdoPR_SUBJECT**.

Example

This code fragment sets the subject of a message:

```
Dim objMessage As Message ' assume valid message
objMessage.Subject = "CDO Library: Check It Out"
```

See Also

Text Property (Message Object)

Submitted Property (Message Object)

Group

The **Submitted** property is **True** when the message has been submitted to the MAPI system. 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 messages, you use the **Sent**, **Submitted**, and **Unread** properties and the **Send** or **Update** methods.

The following table summarizes the use of the message properties and methods for the three kinds of messages. In some systems the message store and transport providers are tightly coupled, in which case they bypass the spooler and perform its functions themselves.

| Kind of message | Method used | Sent property | Submitted property | Unread property |
|-----------------|---------------|-------------------------------|-------------------------------|------------------------------|
| Sent | Send | Spooler sets True | Send sets True | Spooler sets True |
| Posted | Update | Application sets True | Application sets False | Application sets True |
| Saved | Update | Application sets False | Application sets False | Application sets True |

For sent and saved messages, the **Submitted** property is set to **False** before sending or saving the message. The messaging system also resets **Submitted** to **False** when the message arrives in a recipient's Inbox.

For posted messages, you create the message directly within a public folder and call the **Update** method. Some viewers do not allow the message to become visible to other users 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 is not exposed on [AppointmentItem](#) objects created by Microsoft® Schedule+, and it is not automatically set when you create an appointment within a CDO application until you call the **Update** method. In these cases, the application must assign a value to **Submitted** or an attempt to read it returns **CdoE_NOT_FOUND**.

The **Submitted** property corresponds to the MSGFLAG_SUBMIT flag being set in the MAPI property PR_MESSAGE_FLAGS. It can be rendered into HTML hypertext using the CDO Rendering [ObjectRenderer](#) object. To specify this, set the object renderer's **DataSource** property to this Message object and the *property* parameter of the **RenderProperty** method to **CdoPR_MESSAGE_FLAGS**. However, you must assign a CDO Rendering [Format](#) object to the PR_MESSAGE_FLAGS property and use the **Value** property of the format's [Pattern](#) objects to isolate the MSGFLAG_SUBMIT flag's

setting for rendering.

Text Property (Message Object)

Group

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. Text specifically excludes various other message properties such as **Subject**, **Attachments**, and **Recipients**.

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. It can be rendered into HTML hypertext using the CDO Rendering **ObjectRenderer** object. To specify this, set the object renderer's **DataSource** property to this Message object and the *property* parameter of the **RenderProperty** method to **CdoPR_BODY**.

Example

This code fragment sets the text of a message:

```
Dim objMessage As Message ' assume valid message
objMessage.Text = "Thank you for arranging the meeting."
```


TimeCreated Property (Message Object)

Group

The **TimeCreated** property specifies the date/time the message was first saved. Read-only.

Syntax

objMessage.TimeCreated

Data Type

Variant (**vbDate** format)

Remarks

The **TimeCreated** property is not exposed on AppointmentItem objects created by Microsoft® Schedule+, and it is not automatically set when you create an appointment within a CDO application until you call the Update method. In these cases, the application must assign a value to **TimeCreated** or an attempt to read it returns **CdoE_NOT_FOUND**.

The **TimeCreated** property corresponds to the MAPI property PR_CREATION_TIME.

TimeExpired Property (Message Object)

Group

The **TimeExpired** property specifies the date/time the message becomes invalid and can be safely deleted. Read/write.

Syntax

objMessage.TimeExpired

Data Type

Variant (**vbDate** format)

Remarks

The **TimeExpired** property is not required on an [AppointmentItem](#), [MeetingItem](#), or Message object. If the sender does not set **TimeExpired**, an attempt by the receiver to read it returns **CdoE_NOT_FOUND**.

[AppointmentItem](#) objects in a Microsoft® Schedule+ calendar folder do not have the full set of attributes of a general message. If you obtain the default calendar folder by passing **CdoDefaultFolderCalendar** to the [Session](#) object's [GetDefaultFolder](#) method, its appointments have no defined value for the **TimeExpired** property. An attempt to access **TimeExpired** in this case returns **CdoE_NOT_FOUND**.

The **TimeExpired** property corresponds to the MAPI property PR_EXPIRY_TIME.

TimeLastModified Property (Message Object) Group

The **TimeLastModified** property specifies the date/time the message was most recently saved. Read-only.

Syntax

objMessage.TimeLastModified

Data Type

Variant (vbDate format)

Remarks

The **TimeLastModified** property is not exposed on [AppointmentItem](#) objects created by Microsoft® Schedule+, and it is not automatically set when you create an appointment within a CDO application until you call the [Update](#) method. In these cases, the application must assign a value to **TimeLastModified** or an attempt to read it returns **CdoE_NOT_FOUND**.

The **TimeLastModified** property corresponds to the MAPI property PR_LAST_MODIFICATION_TIME.

TimeReceived Property (Message Object) Group

The **TimeReceived** property sets or returns the date/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.

The **TimeReceived** and **TimeSent** properties represent local time.

AppointmentItem objects in a Microsoft® Schedule+ calendar folder do not have the full set of attributes of a general message. If you obtain the default calendar folder by passing **CdoDefaultFolderCalendar** to the Session object's **GetDefaultFolder** method, its appointments have no defined value for the **TimeReceived** property. An attempt to access **TimeReceived** in this case returns **CdoE_NOT_FOUND**.

The **TimeReceived** property corresponds to the MAPI property PR_MESSAGE_DELIVERY_TIME. It can be rendered into HTML hypertext using the CDO Rendering ObjectRenderer object. To specify this, set the object renderer's **DataSource** property to this Message object and the *property* parameter of the RenderProperty method to **CdoPR_MESSAGE_DELIVERY_TIME**.

Note When PR_MESSAGE_DELIVERY_TIME is rendered into HTML, it is converted to the local time at IIS, not the browser's local time.

Example

This code fragment displays the date/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)

Group

The **TimeSent** property sets or returns the date/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.

The **TimeReceived** and **TimeSent** properties represent local time.

AppointmentItem objects in a Microsoft® Schedule+ calendar folder do not have the full set of attributes of a general message. If you obtain the default calendar folder by passing **CdoDefaultFolderCalendar** to the **Session** object's **GetDefaultFolder** method, its appointments have no defined value for the **TimeSent** property. An attempt to access **TimeSent** in this case returns **CdoE_NOT_FOUND**.

The **TimeSent** property corresponds to the MAPI property PR_CLIENT_SUBMIT_TIME. It can be rendered into HTML hypertext using the CDO Rendering **ObjectRenderer** object. To specify this, set the object renderer's **DataSource** property to this Message object and the *property* parameter of the **RenderProperty** method to **CdoPR_CLIENT_SUBMIT_TIME**.

Note When PR_CLIENT_SUBMIT_TIME is rendered into HTML, it is converted to the local time at IIS, not the browser's local time.

Example

This code fragment displays the date/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)

Group

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.

The subclasses of the Message object are distinguished by the value of their **Type** property. An [AppointmentItem](#) object has a **Type** of IPM.Appointment, and a [MeetingItem](#) object has a **Type** of IPM.Schedule.Meeting.Request.

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

CDO 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, CDO sets the **Type** value of new messages to the MAPI message class IPM.Note.

[AppointmentItem](#) objects in a Microsoft® Schedule+ calendar folder do not have the full set of attributes of a general message. If you obtain the default calendar folder by passing **CdoDefaultFolderCalendar** to the [Session](#) object's [GetDefaultFolder](#) method, its appointments have no defined value for the **Type** property. An attempt to access **Type** in this case returns **CdoE_NOT_FOUND**.

The **Type** property corresponds to the MAPI property PR_MESSAGE_CLASS. It can be rendered into HTML hypertext using the CDO Rendering [ObjectRenderer](#) object. To specify this, set the object renderer's **DataSource** property to this Message object and the *property* parameter of the [RenderProperty](#) method to **CdoPR_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 messages, you use the **Sent**, **Submitted**, and **Unread** properties and the **Send** or **Update** methods.

The following table summarizes the use of the message properties and methods for the three kinds of messages. In some systems the message store and transport providers are tightly coupled, in which case they bypass the spooler and perform its functions themselves.

| Kind of message | Method used | Sent property | Submitted property | Unread property |
|-----------------|---------------|-------------------------------|-------------------------------|------------------------------|
| Sent | Send | Spooler sets True | Send sets True | Spooler sets True |
| Posted | Update | Application sets True | Application sets False | Application sets True |
| Saved | Update | Application sets False | Application sets False | Application sets True |

The **Unread** property should initially be **True** for all three kinds of messages. The messaging system takes care of this for a sent message; you must set the property for a posted or saved message. The messaging system also resets **Submitted** to **False** when the message arrives in a recipient's Inbox. Each receiving user sets the **Unread** property to **False** as it reads its copy of the received message.

Note When you set **Unread** to **False**, the MSGFLAG_READ flag of the MAPI property PR_MESSAGE_FLAGS is immediately updated in the message store. However, to improve performance by reducing RPCs, the store might not communicate this change to a messaging client until contacted by the client. Therefore, a client such as the Microsoft® Exchange Client or Microsoft® Outlook™ may continue to show the message as unread for some indeterminate period of time.

For more information on sending and posting messages, see [Creating and Sending a Message and Posting Messages to a Public Folder](#).

The **Unread** property is not exposed on [AppointmentItem](#) objects created by Microsoft® Schedule+, and it is not automatically set when you create an appointment within a CDO application until you call the **Update** method. In these cases, the application must assign a value to **Unread** or an attempt to read it returns **CdoE_NOT_FOUND**.

The **Unread** property corresponds to the MSGFLAG_READ flag not being set in the MAPI property PR_MESSAGE_FLAGS. It can be rendered into HTML hypertext using the CDO Rendering [ObjectRenderer](#) object. To specify this, set the object renderer's **DataSource** property to this Message

object and the *property* parameter of the RenderProperty method to **CdoPR_MESSAGE_FLAGS**. However, you must assign a CDO Rendering Format object to the PR_MESSAGE_FLAGS property and use the Value property of the format's Pattern objects to isolate the MSGFLAG_READ flag's setting for rendering.

Update Method (Message Object)

Group

The **Update** method saves the message in the MAPI system.

Syntax

```
objMessage.Update( [makePermanent] [, refreshObject] )
```

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 a Message object are not permanently saved in the MAPI system until you either call the **Update** method with the *makePermanent* parameter set to **True** or call the **Send** method.

For improved performance, CDO 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 Message ' Message object  
'  
  
Set objMessage = objSession.Outbox.Messages.Add  
objMessage.Subject = "Microsoft CDO Library"  
objMessage.Text = "The new version is now available."
```

objMessage.Update makePermanent:=True ' redundant parameter (default)

MessageFilter Object

The MessageFilter object specifies criteria for restricting a search on a [Messages](#) collection.

At a Glance

| | |
|----------------------------|-------------------------------------|
| Specified in type library: | CDO.DLL |
| First available in: | CDO Library version 1.1 |
| Parent objects: | Messages collection |
| Child objects: | Fields collection |
| Default property: | Subject |

Properties

| Name | Available since 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 | String | Read/write |
| Sender | 1.1 | String | 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 since version | Parameters |
|---------------------------------|-------------------------|---------------------------------------|
| IsSameAs | 1.1 | <i>objMsgFilter2</i> 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 or adding fields to its Fields collection. You do not need to call any **Update** method when setting filter criteria.

The filter is invoked when the Messages collection is traversed with the **Get** methods or the Microsoft® Visual Basic® **For Each** construction. Each field participates in a MAPI search restriction comparing the field's Value property against the value of the Message object's property specified by the field's ID property.

When you are traversing a Messages collection instantiated by a CDO rendering application, you should declare the Visual Basic variable to be an **Object** instead of a Message. This is because the collection may be from a calendar folder, and also because a rendering may have applied a grouped view to the folder. Therefore you can get AppointmentItem, GroupHeader, and MeetingItem objects returned as well as Message objects. You should also test the Class property of each returned object to see if it is an appointment, a group header, a meeting, or a message:

```
Dim objMember As Object ' could get one of several classes
' collMessages is instantiated from a rendering application
' assume collMessages valid
' ...
For Each objMember in collMessages ' collection from a rendering
  If objMember.Class = CdoMsg ' exclude other classes
    ' we have a Message object
  End If
Next
```

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. However, the following MAPI properties are compared using FL_PREFIX instead of FL_SUBSTRING:

```
PR_ACCOUNT
PR_BUSINESS_ADDRESS_CITY
PR_COMPANY_NAME
PR_DEPARTMENT_NAME
PR_DISPLAY_NAME
PR_GIVEN_NAME
PR_OFFICE_LOCATION
PR_SURNAME
PR_TITLE
```

If the underlying messaging system does not support the search criteria specified by the filter fields, the **Get** methods return **CdoE_TOO_COMPLEX**. In particular, a given field may not be supported by a particular message store.

You can apply a MessageFilter object to a Messages collection containing AppointmentItem objects. However, the current version of CDO only supports filtering on the appointment's EndTime and StartTime properties. An attempt to filter on any other properties, including the inherited Message object properties, returns **CdoE_TOO_COMPLEX**. The current version of CDO also does not support any of the MessageFilter's properties other than the Fields property when the filter is applied to a collection of appointments.

MeetingItem objects are members of a Messages collection belonging to a messaging user's Inbox, and they can be filtered on any properties, including the inherited Message object properties.

The MAPI search restrictions for appointment properties are as follows:

| Property tag | Restriction type | Relational operator |
|-------------------------|------------------|---------------------|
| CdoPR_END_DATE | RES_PROPERTY | RELOP_GE |
| CdoPR_START_DATE | RES_PROPERTY | RELOP_LE |

The comparison order is (*property value*) (*relational operator*) (*constant value*). For example, the message filter passes appointments with a starting date earlier than or equal to the date you indicate. To specify this restriction, you use the filter's **Fields** property to obtain its Fields collection, **Add** a new Field with **CdoPR_START_DATE** in the *PropTag* parameter, and set the new field's **Value** property to the last admissible starting date.

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

The message filter affects traversals of the Messages collection using the Visual Basic **For Each** statement, the **Get** methods, or the Visual Basic **For ... Next** construction. These accesses normally return a Message object but can also return a GroupHeader object if the collection is instantiated by a CDO rendering application.

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. You can use the following code to clear a message filter of all of its previous settings and reset it to its default state of no criteria:

```
objMessagesColl.Filter = Nothing ' filter now invalidated and cleared  
Set objMessageFilt = objMessagesColl.Filter ' new valid empty filter
```

If a folder is being rendered with a CDO rendering application, the Messages collection and the message filter are instantiated according to the specifications in the TableView object being applied to the folder. The MessageFilter object inherits the restriction specified in the view. An inherited filter can be used without modification, but it cannot be read or changed by the rendering application. Writing any property on an inherited filter disinherits it and refreshes the Messages collection. This means that the collection is reinstantiated with a new message filter specifying only the property just written. This new filter, however, is no longer inherited, and the application can read its properties and set additional restrictions within it.

Example

This code fragment specifies that the message filter on the Inbox should pass only Message objects that are **Not** of message class IPM.Note **AND** are **Unread**:

```
Dim objMsgFilt As MessageFilter  
...  
Set objMsgFilt = objSession.Inbox.Messages.Filter  
' ... validate MessageFilter object ...  
objMsgFilt.Not = True ' negate all results before ANDing  
objMsgFilt.Type = "IPM.Note" ' MAPI message class  
objMsgFilt.Unread = False ' .Not setting negates True to False!
```

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 Message object's **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)

Group

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

Integer. Value must be greater than 0 and less than or equal to 65,535 (&HFFFF). Specifies the index within the Fields collection.

proptag

Long. Value must be greater than or equal to 65,536 (&H10000). 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 (Field or Fields collection)

Remarks

The **Fields** property returns one or all of the fields associated with a MessageFilter object. Each field typically corresponds to a MAPI property. Data types are preserved, except that MAPI counted binary properties are converted to and from character strings representing hexadecimal digits.

The fields that have been set in the [Fields](#) collection specify the filter, together with any other MessageFilter properties that have been set.

The **Fields** property provides a generic access mechanism that allows Microsoft® Visual Basic® and Microsoft® 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 **CdoPR_MESSAGE_FLAGS**. To access a named property, use *objMessageFilter.Fields*(*name*), where *name* is a string that represents the custom property name.

Note Not all message stores support filtering on all possible fields, that is, on all possible MAPI properties. A store typically returns **CdoE_TOO_COMPLEX** if you specify a property on which it does not support filtering.

Although the **Fields** property itself is read-only, the collection it returns can be accessed in the normal manner through its **Add** and **Delete** methods, and the properties on its member [Field](#) objects retain their respective read/write or read-only accessibility.

Importance Property (MessageFilter Object) Group

The **Importance** property sets filtering on a message's importance to **CdoNormal** (the default), **CdoLow**, or **CdoHigh**. Read/write.

Syntax

objMessageFilter.Importance

Data Type

Long

Remarks

The following values are defined:

| Constant | Value | Description |
|------------------|-------|-----------------------------|
| CdoLow | 0 | Low importance |
| CdoNormal | 1 | Normal importance (default) |
| CdoHigh | 2 | High importance |

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

IsSameAs Method (MessageFilter Object)

Group

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

Syntax

```
boolSame = objMessageFilter.IsSameAs(objMsgFilter2)
```

boolSame

On successful return, contains **True** if the two objects are the same.

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 in the underlying messaging system. 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. This is necessary because, although MAPI requires all entry identifiers to be unique, it does not require two of them identifying the same object to be identical. A generic comparison of any two objects' unique identifiers is also available with the Session object's **CompareIDs** method.

Not Property (MessageFilter Object)

Group

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)

Group

The **Or** property specifies that the restriction values are to be **ORed** instead of **ANDed** 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 **ANDed** together. If it is **True**, the values are **ORed** together.

Recipients Property (MessageFilter Object)

The **Recipients** property sets filtering on whether a message's recipients include at least one recipient with a particular name. Read/write.

Syntax

objMessageFilter.**Recipients**

Data Type

String

Remarks

The **Recipients** property specifies that the message filter should pass only messages with one or more recipients having a name corresponding to the **Recipients** property. The filter passes the message if the **Name** property of any of its Recipient objects contains the filter's **Recipients** 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 Recipient
' assume objMessage and objMessageFilter are valid
For i = 1 To objMessage.Recipients.Count Step 1
    strRecipName = objMessage.Recipients.Item(i).Name
' or objMessage.Recipients(i) since Item and Name are default properties
    If strRecipName <> "" Then
        objMessageFilter.Recipients = strRecipName
        Exit For
    End If
Next i
```

Sender Property (MessageFilter Object)

Group

The **Sender** property sets filtering on the name of a message's sender. Read/write.

Syntax

objMessageFilter.Sender

Data Type

String

Remarks

The **Sender** property specifies that the message filter should pass only messages sent by a messaging user having a name corresponding to the **Sender** property. The filter passes the message if the **Name** property of the AddressEntry object returned by the message's **Sender** property contains the filter's **Sender** property as a substring.

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

Sent Property (MessageFilter Object)

Group

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)

Group

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)

Group

The **Subject** property sets filtering on a message's subject. Read/write.

Syntax

objMessageFilter.**Subject**

The **Subject** property is the default property of a MessageFilter object, meaning that *objMessageFilter* is syntactically equivalent to *objMessageFilter*.**Subject** in Microsoft® Visual Basic® code.

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.

Text Property (MessageFilter Object)

Group

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) Group

The **TimeFirst** property sets filtering on whether a message was received at or since the specified date/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/time in the **TimeLast** property. If neither property is set, the filter passes messages regardless of their date/time of reception.

The **TimeFirst** and **TimeLast** properties represent local time.

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

TimeLast Property (MessageFilter Object) Group

The **TimeLast** property sets filtering on whether a message was received at or before the specified date/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/time in the **TimeFirst** property. If neither property is set, the filter passes messages regardless of their date/time of reception.

For more information and an example using the **TimeLast** property, see [Filtering Messages in a Folder](#).

The **TimeFirst** and **TimeLast** properties represent local time.

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

Type Property (MessageFilter Object)

Group

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, CDO 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)

Group

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 AppointmentItem, GroupHeader, MeetingItem, and Message objects.

At a Glance

| | |
|----------------------------|---|
| Specified in type library: | CDO.DLL |
| First available in: | CDO Library version 1.0.a |
| Parent objects: | Folder |
| Child objects: | AppointmentItem
GroupHeader
MeetingItem
Message
MessageFilter |
| Default property: | Item |

A Messages collection is considered a *large collection*, which means that the **Count** property has limited validity, and the best way to access an individual GroupHeader or Message object within the collection is to use either its unique identifier or the **Get** methods. For more information on collections, see [Object Collections](#).

Properties

| Name | Available since 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 | GroupHeader object or Message object | Read-only |
| Parent | 1.0.a | Folder object | Read-only |
| RawTable | 1.1 | IUnknown object | Read/write
(Note: Not available to Visual Basic applications) |
| Session | 1.0.a | Session object | Read-only |

Methods

| Name | Available since version | Parameters |
|---------------------|-------------------------|---|
| Add | 1.0.a | (optional) <i>subject</i> as String ,
(optional) <i>text</i> as String ,
(optional) <i>type</i> as String ,
(optional) <i>importance</i> as Long |

| | | |
|---------------------------|-------|---|
| <u>Delete</u> | 1.0.a | (none) |
| <u>GetFirst</u> | 1.0.a | (optional) <i>filter</i> as String |
| <u>GetLast</u> | 1.0.a | (optional) <i>filter</i> as String |
| <u>GetNext</u> | 1.0.a | (none) |
| <u>GetPrevious</u> | 1.0.a | (none) |
| <u>Sort</u> | 1.0.a | (optional) <i>SortOrder</i> as Long ,
(optional) <i>PropTag</i> as Long ,
(optional) <i>PropID</i> as String |

Remarks

A Messages collection can be rendered into HTML hypertext in tabular form using the CDO Rendering [ContainerRenderer](#) object. To specify this, set the container renderer's **DataSource** property to the Messages collection object itself.

With the same **DataSource** setting, the container renderer's **RenderProperty** method can also render selected properties of the collection's parent [Folder](#) object. The individual properties that can be rendered are indicated in the Folder object property descriptions.

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**, **GetLast**, **GetNext**, and **GetPrevious** methods to access individual items in the collection. You can access one specific appointment, meeting, or 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**, **GetLast**, **GetNext**, and **GetPrevious** depends on whether they are sorted or not. The [AppointmentItem](#), [GroupHeader](#), [MeetingItem](#), and [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.

[GroupHeader](#) objects are not saved in the message store. They cannot be created or deleted programmatically by your application. They are generated only when a CDO Rendering [TableView](#) object is being applied to the Messages collection, and they do not persist when the view is released.

Add Method (Messages Collection)

Group

The **Add** method creates and returns a new [AppointmentItem](#) or [Message](#) object in the Messages collection.

Syntax

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

objMessage

On successful return, represents the new [AppointmentItem](#) or [Message](#) object added to the collection. The type of object added depends on the parent folder of the Messages 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 |
|------------------|-------|-----------------------------|
| CdoLow | 0 | Low importance |
| CdoNormal | 1 | Normal importance (default) |
| CdoHigh | 2 | High importance |

Remarks

The method parameters correspond to the [Subject](#), [Text](#), [Type](#), and [Importance](#) properties of the [Message](#) object.

Note If you are adding an [AppointmentItem](#) object to a calendar folder, you cannot use any of the parameters of the **Add** method. You can, however, set the values later by using the corresponding properties.

You should create new messages in the Inbox or Outbox folder, and new appointments in the calendar 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 replies to an original message:

```
' from the sample function Util_ReplyToConversation
Set objNewMsg = objSession.Outbox.Messages.Add
' verify objNewMsg created successfully ... then supply properties
Set objSenderAE = objOriginalMsg.Sender ' sender as AddressEntry
```



```
With objNewMsg
    .Text = "How about a slightly used bicycle?" ' new text
    .Subject = objOriginalMsg.Subject ' copy original properties
    .ConversationTopic = objOriginalMsg.ConversationTopic
    ' append time stamp; compatible with Microsoft Exchange client
    Set objOneRecip = .Recipients.Add( _
        Name:=objSenderAE.Name, _
        Address:=objSenderAE.Type & ":" & objSenderAE.Address, _
        Type:=CdoTo)
    .Recipients.Resolve
    .Update
    .Send showDialog:=False
End With
```

Count Property (Messages Collection)

Group

The **Count** property returns the number of [AppointmentItem](#), [MeetingItem](#), or [GroupHeader](#) and [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

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 the value &H7FFFFFFF. Programmers needing to access individual objects in a large collection are strongly advised to use the Microsoft® 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 message store provider cannot supply the precise number of objects, CDO returns &H7FFFFFFF ($= 2^{31} - 1 = 2,147,483,647$) for the **Count** property. This is the largest positive value for a long integer and is intended to prevent an approximate count from prematurely terminating an indexed loop. On 32-bit platforms, this value is defined in the type library as **CdoMaxCount**. On other platforms, **CdoMaxCount** is not defined, and a program on such a platform must compare the **Count** property against &H7FFFFFFF to see if it is reliable.

If the **Count** property is not reliable, that is, if it is &H7FFFFFFF, a program using it to terminate an indexed loop must also check each returned object for a value of **Nothing** to avoid going past the end of the collection.

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". Note that the variable is declared as **Object** instead of **Message**, and that the **Class** property is tested to verify that the object returned in the **Item** property is not an [AppointmentItem](#), [GroupHeader](#), or [MeetingItem](#) object.

```
Dim i As Integer ' loop index / object counter
Dim collMessages As Messages ' assume collection already provided
Dim objMsg As Object ' could get either group header or message
If collMessages Is Nothing Then
    MsgBox "Messages collection object is invalid"
    ' Exit
Elseif 0 = collMessages.Count Then ' collection is empty
    MsgBox "No messages in collection"
    ' Exit
End If
' look for message about "Bonus" in collection
For i = 1 To collMessages.Count Step 1
```

```
Set objMsg = collMessages.Item(i)
' or collMessages(i) since Item is default property
If objMsg Is Nothing Then ' end of collection
    MsgBox "No such message found in collection"
    Exit For
Elseif objMsg.Class = CdoMsg Then ' exclude other object classes
    If 0 = StrComp(objMsg.Subject, "Bonus") Then
        ' or objMsg since Subject is default property
        MsgBox "Desired message is at index " & i
        Exit For
    End If
End If
Next i
```

Delete Method (Messages Collection)

Group

The **Delete** method removes all the [AppointmentItem](#), [MeetingItem](#), [GroupHeader](#), and [Message](#) objects from the Messages collection.

Syntax

objMsgColl.Delete()

Remarks

The **Delete** method performs an irreversible operation on the collection. It calls **Release** on the collection's reference to every [AppointmentItem](#), [GroupHeader](#), [MeetingItem](#), and [Message](#) object. If you have another reference to an object, you can still access its properties and methods, but you can never again associate it with any collection because the **Add** method always creates a new object. You should **Set** your reference variable either to **Nothing** or to another object.

The final **Release** on each [AppointmentItem](#), [GroupHeader](#), [MeetingItem](#), or [Message](#) object takes place when you assign your reference variable to **Nothing**, or when you call **Delete** if you had no other reference. At this point the object is removed from memory. Attempted access to a released object results in an error return of **CdoE_INVALID_OBJECT**.

The **Delete** method does not move any [MeetingItem](#) or [Message](#) objects to the Deleted Items folder. If this folder exists and you wish this functionality, you must specifically program your application for it.

If the Messages collection underlies a categorized view, the [GroupHeader](#) objects are also removed from the collection, but unlike the messages they are not moved anywhere. Group headers do not persist in storage, and when the collection is released, whether with messages or not, the [GroupHeader](#) objects cease to exist.

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 is still valid, and you can **Add** new members to it.

Filter Property (Messages Collection)

Group

The **Filter** property returns a [MessageFilter](#) object for the Messages collection. Read/write.

Syntax

objMsgColl.Filter

Data Type

Object (MessageFilter)

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](#).

A message filter can also be inherited from the restriction specified in a CDO Rendering [TableView](#) object. Writing any property on this filter disinherits it, refreshes the Messages collection, and instantiates a new message filter specifying only the property just written. This new filter, however, is no longer inherited, and the application can read its properties and set additional restrictions within it.

The message filter affects traversals of the Messages collection using the Microsoft® Visual Basic® **For Each** statement, the **Get** methods, or the Visual Basic **For ... Next** construction. These accesses normally return a [Message](#) object but can also return an [AppointmentItem](#) object if the collection resides in a calendar folder, a [GroupHeader](#) object if the collection is instantiated by a CDO rendering application, or a [MeetingItem](#) object if the collection is in an Inbox or Outbox.

Example

This code fragment shows how to set a filtering value in a Messages collection's initial default message filter, and then how to clear all settings and reset the filter to its default state of no criteria:

```
Dim objMsgColl As Messages ' collection
Dim objMessage As Message ' message passed by filter
Dim objMsgFilt As MessageFilter
' assume valid Messages collection just created
' make first use of filter to check for high importance messages
Set objMsgFilt = objMsgColl.Filter ' original empty default filter
objMsgFilt.Importance = CdoHigh
For Each objMessage in objMsgColl ' loops and Sets each objMessage
    ' process messages that are passed by the filter
Next
' ... later, when current filter settings are no longer needed ...
objMsgColl.Filter = Nothing ' invalidates and clears filter
Set objMsgFilt = objMsgColl.Filter ' new empty filter
' filter now available for new settings
```

GetFirst Method (Messages Collection)

Group

The **GetFirst** method returns the first [AppointmentItem](#), [GroupHeader](#), [MeetingItem](#), or [Message](#) object in the Messages collection. It returns **Nothing** if no first object exists.

Syntax

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

objMessage

On successful return, represents the first AppointmentItem, GroupHeader, MeetingItem, or Message object in the collection.

objMsgColl

Required. The Messages collection object.

filter

Optional. String. Specifies the message class of the object, such as IPM.Note, the default value. Corresponds to the [Type](#) property of the Message object.

Remarks

If the *filter* parameter is set, the **GetFirst** method returns the first meeting or message in the collection with a [Type](#) property matching the value of *filter*. However, if the Messages collection is in a calendar folder, the *filter* parameter is not supported. Microsoft® Outlook™ ignores any setting of *filter* and returns the first appointment, while Microsoft® Schedule+ returns **CdoE_TOO_COMPLEX** if *filter* is set.

The **Get** methods normally return a [Message](#) object but can also return an [AppointmentItem](#), [GroupHeader](#), or [MeetingItem](#) object.

The order that items are returned by **GetFirst**, **GetLast**, **GetNext**, and **GetPrevious** depends on whether they are sorted or not. The AppointmentItem, GroupHeader, MeetingItem, and 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.

The **Get** methods sometimes perform more efficiently if you have applied a [MessageFilter](#) object to the collection.

GetLast Method (Messages Collection)

Group

The **GetLast** method returns the last [AppointmentItem](#), [GroupHeader](#), [MeetingItem](#), or [Message](#) object in the Messages collection. It returns **Nothing** if no last object exists.

Syntax

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

objMessage

On successful return, represents the last [AppointmentItem](#), [GroupHeader](#), [MeetingItem](#), or [Message](#) object in the collection.

objMsgColl

Required. The Messages collection object.

filter

Optional. String. Specifies the message class of the object, such as IPM.Note, the default value. Corresponds to the [Type](#) property of the Message object.

Remarks

If the *filter* parameter is set, the **GetLast** method returns the last message in the collection with a [Type](#) property matching the value of *filter*. However, if the Messages collection is in a calendar folder, the *filter* parameter is not supported. Microsoft® Outlook™ ignores any setting of *filter* and returns the last appointment, while Microsoft® Schedule+ returns **CdoE_TOO_COMPLEX** if *filter* is set.

The **Get** methods normally return a [Message](#) object but can also return an [AppointmentItem](#), [GroupHeader](#), or [MeetingItem](#) object.

The order that items are returned by [GetFirst](#), [GetLast](#), [GetNext](#), and [GetPrevious](#) depends on whether they are sorted or not. The [AppointmentItem](#), [GroupHeader](#), [MeetingItem](#), and [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.

The **Get** methods sometimes perform more efficiently if you have applied a [MessageFilter](#) object to the collection.

If a Messages collection has not been enumerated since it was initialized, the behavior of the **GetLast** method is not defined. A collection is enumerated when you call its [GetFirst](#) method, access a member through its [Item](#) property, or use its [Filter](#) property to specify a message filter. Some providers may return **CdoE_CALL_FAILED** if you call **GetLast** before the collection is enumerated.

Microsoft Schedule+ does not support **GetLast** on a Messages collection of [AppointmentItem](#) objects.

GetNext Method (Messages Collection)

Group

The **GetNext** method returns the next [AppointmentItem](#), [GroupHeader](#), [MeetingItem](#), or [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**()

objMessage

On successful return, represents the next [AppointmentItem](#), [GroupHeader](#), [MeetingItem](#), or [Message](#) object in the collection.

objMsgColl

Required. The Messages collection object.

Remarks

The **Get** methods normally return a [Message](#) object but can also return an [AppointmentItem](#), [GroupHeader](#), or [MeetingItem](#) object.

The order that items are returned by **GetFirst**, **GetLast**, **GetNext**, and **GetPrevious** depends on whether they are sorted or not. The [AppointmentItem](#), [GroupHeader](#), [MeetingItem](#), and [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.

The **Get** methods sometimes perform more efficiently if you have applied a [MessageFilter](#) object to the collection.

If a Messages collection has not been enumerated since it was initialized, the behavior of the **GetNext** method is not defined. A collection is enumerated when you call its **GetFirst** method, access a member through its **Item** property, or use its **Filter** property to specify a message filter. Some providers may return **CdoE_CALL_FAILED** if you call **GetNext** before the collection is enumerated.

Calling **GetNext** on an unenumerated collection can produce unexpected results if the collection is reinitialized with a **Set** statement in every iteration of a loop. The recommended procedure is to **Set** an explicit variable for the collection before entering the loop. For more information, see [Object Collections](#).

GetPrevious Method (Messages Collection) Group

The **GetPrevious** method returns the previous [AppointmentItem](#), [GroupHeader](#), [MeetingItem](#), or [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( )
```

objMessage

On successful return, represents the previous [AppointmentItem](#), [GroupHeader](#), [MeetingItem](#), or [Message](#) object in the collection.

objMsgColl

Required. The Messages collection object.

Remarks

The **Get** methods normally return a [Message](#) object but can also return an [AppointmentItem](#), [GroupHeader](#), or [MeetingItem](#) object.

The order that items are returned by **GetFirst**, **GetLast**, **GetNext**, and **GetPrevious** depends on whether they are sorted or not. The [AppointmentItem](#), [GroupHeader](#), [MeetingItem](#), and [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.

The **Get** methods sometimes perform more efficiently if you have applied a [MessageFilter](#) object to the collection.

If a Messages collection has not been enumerated since it was initialized, the behavior of the **GetPrevious** method is not defined. A collection is enumerated when you call its **GetFirst** method, access a member through its [Item](#) property, or use its [Filter](#) property to specify a message filter. Some providers may return **CdoE_CALL_FAILED** if you call **GetPrevious** before the collection is enumerated.

Calling **GetPrevious** on an unenumerated collection can produce unexpected results if the collection is reinitialized with a **Set** statement in every iteration of a loop. The recommended procedure is to **Set** an explicit variable for the collection before entering the loop. For more information, see [Object Collections](#).

Microsoft Schedule+ does not support **GetPrevious** on a Messages collection of [AppointmentItem](#) objects.

Item Property (Messages Collection)

Group

The **Item** property returns a single [AppointmentItem](#), [GroupHeader](#), [MeetingItem](#), or [Message](#) object from the Messages collection. Read-only.

Syntax

objMsgColl.**Item**(*index*)

objMsgColl.**Item**(*searchValue*)

index

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

searchValue

A string used to search the Messages collection starting at the current position. The search returns the next Message object having the current sorting property greater than or equal to the *searchValue* string.

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

Data Type

Object (GroupHeader or Message)

Remarks

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 returns the AppointmentItem, GroupHeader, MeetingItem, or Message object at the indicated position in the collection. It can be used in an indexed loop, such as the **For ... Next** construction in Visual Basic. The first item in the collection has an index of 1.

If you are accessing a Messages collection instantiated by a CDO rendering application, the collection may come from a calendar folder, or there may be a grouped view applied to the folder. Therefore you can get [AppointmentItem](#), [GroupHeader](#), and [MeetingItem](#) objects returned as well as [Message](#) objects. Because of this, you should declare the Visual Basic variable being set to the **Item** property to be an **Object** rather than a Message, and you should also test the **Class** property of each returned object to see if it is an appointment, group header, meeting, or message:

```
Dim objMember As Object ' could get one of several classes
' collMessages is instantiated from a rendering application
' assume collMessages valid
' ...
For Each objMember in collMessages ' collection from a rendering
    If objMember.Class = CdoMsg ' exclude other classes
        ' we have a Message object
    End If
Next
```

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

The **Item**(*searchValue*) syntax returns the next Message object whose current sorting property is greater than or equal to the string specified by *searchValue*. This syntax only applies when the Messages collection contains Message objects.

The *searchValue* syntax starts its search at the current position and retrieves only messages and not group headers. Searching is based on the current sort order of the collection. The default sort property for a Messages collection is the **TimeReceived** property of the collection's Message objects. If you want to use the **Item(searchValue)** syntax to search the collection on another property, for example a message subject, you should first call the **Sort** method specifying the **Subject** property.

Note The **Item(searchValue)** syntax uses the **IMAPITABLE::FindRow** method, which performs a search dependent on the current sort order of the table underlying the collection. Not all tables are sorted alphabetically. If your most recent sort order is nonalphabetic, you should access the messages using the **Item(index)** syntax. This applies, for example, to messages in Microsoft Exchange Public Folders, which are held in an order determined by the currently applied view.

For more information on tables, bookmarks, restrictions, and sort and search orders, see the *MAPI Programmer's Reference*.

Although the **Item** property itself is read-only, the AppointmentItem, GroupHeader, MeetingItem, or Message object it returns can be accessed in the normal manner, and its properties retain their respective read/write or read-only accessibility.

RawTable Property (Messages Collection) Group

The **RawTable** property returns an **IUnknown** pointer to the MAPI table object underlying the Messages collection. Not available to Microsoft® Visual Basic® applications. Read/write.

Syntax

objMsgColl.RawTable

Data Type

Variant (**vbDataObject** format)

Remarks

The **RawTable** property is not available to Visual Basic programs. It is accessible only by C/C++ programs that deal with **IUnknown** objects. Visual Basic supports the **IDispatch** interface and not **IUnknown**. The **RawTable** property is an **IUnknown** object that returns an **IMAPITable** interface in response to **QueryInterface**. For more information, see [Introduction to Automation](#) and [How Programmable Objects Work](#). Also see the "COM and ActiveX Object Services" section of the Microsoft Platform SDK.

If your application uses any **MAPIOBJECT** or **RawTable** properties, it must **Release** them all before calling the [Session](#) object's **Logoff** method. Failure to do so can result in unexpected behavior.

The **RawTable** property does not correspond to a MAPI property and cannot be rendered into HTML hypertext by the CDO Rendering Library.

Sort Method (Messages Collection)

Group

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

Syntax

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

objMsgColl.**Sort**([*SortOrder*] [, *name*])

objMsgColl

Required. The Messages collection object.

SortOrder

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

| Value | Numeric value | Description |
|----------------------|---------------|--------------------------|
| CdoNone | 0 | No sort |
| CdoAscending | 1 | Ascending sort (default) |
| CdoDescending | 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 **CdoPR_MESSAGE_CLASS**.

name

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 *name* 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 **CdoPR_MESSAGE_DELIVERY_TIME** is used for the sort.

Each call to **Sort** generates an entirely new sort order based on the specified property. No previous sort order is retained or nested.

If the Messages collection was instantiated by a CDO rendering application, a view has been applied to the folder and the collection is already sorted on the property or properties specified by the view.

If the Messages collection is in a calendar folder, the [AppointmentItem](#) objects are already sorted on **CdoPR_START_DATE** and cannot be sorted on any other property. An attempt do so when running with Microsoft® Outlook™ returns **CdoE_TOO_COMPLEX**. Any call to **Sort** in a calendar folder when running with Microsoft Schedule+ returns **CdoE_NO_SUPPORT**. [MeetingItem](#) objects, however, can be sorted just like [Message](#) objects.

If the underlying messaging system does not support the sort criteria specified, for example descending order or MAPI named properties, the **Sort** method returns **CdoE_TOO_COMPLEX**.

Recipient Object

The Recipient object represents a recipient of a message.

At a Glance

| | |
|----------------------------|------------------------------|
| Specified in type library: | CDO.DLL |
| First available in: | CDO Library version 1.0.a |
| Parent objects: | <u>Recipients</u> collection |
| Child objects: | <u>AddressEntry</u> |
| Default property: | <u>Name</u> |

Properties

| Name | Available since version | Type | Access |
|------------------------------|-------------------------|----------------------------------|------------|
| <u>Address</u> | 1.0.a | String | Read/write |
| <u>AddressEntry</u> | 1.0.a | AddressEntry object | Read/write |
| <u>AmbiguousNames</u> | 1.1 | AddressEntries collection object | Read-only |
| <u>Application</u> | 1.0.a | String | Read-only |
| <u>Class</u> | 1.0.a | Long | Read-only |
| <u>DisplayType</u> | 1.0.a | Long | Read-only |
| <u>ID</u> | 1.1 | String | Read/write |
| <u>Index</u> | 1.0.a | Long | Read-only |
| <u>MeetingResponseStatus</u> | 1.2 | Long | Read/write |
| <u>Name</u> | 1.0.a | String | Read/write |
| <u>Parent</u> | 1.0.a | Recipients collection object | Read-only |
| <u>Session</u> | 1.0.a | Session object | Read-only |
| <u>Type</u> | 1.0.a | Long | Read/write |

Methods

| Name | Available since version | Parameters |
|--------------------|-------------------------|--|
| <u>Delete</u> | 1.0.a | (none) |
| <u>GetFreeBusy</u> | 1.2 | <i>StartTime</i> as Variant ,
<i>EndTime</i> as Variant ,
<i>Interval</i> as Long |
| <u>IsSameAs</u> | 1.1 | <i>objRecip2</i> as Object |
| <u>Resolve</u> | 1.0.a | (optional) <i>showDialog</i> as Boolean |

Address Property (Recipient Object)

Group

The **Address** property specifies the *full address* for the recipient. Read/write.

Syntax

objRecipient.Address

Data Type

String

Remarks

You should use 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.

CDO 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 a combination of 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) Group

The **AddressEntry** property contains the [AddressEntry](#) object representing the recipient. Read/write.

Syntax

objRecipient.AddressEntry

Data Type

Object (AddressEntry)

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, CDO reports an error. For example, when the recipient contains an empty string, the resolve operation returns **CdoE_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:
```


AmbiguousNames Property (Recipient Object) Group

The **AmbiguousNames** property returns an [AddressEntries](#) collection of suggestions for address resolution of the Recipient object. Read-only.

Syntax

objRecipient.**AmbiguousNames**

Data Type

Object (AddressEntries collection)

Remarks

The **AmbiguousNames** property is used when the recipient has not been resolved by the Recipient object's **Resolve** method or the [Recipients](#) collection's **Resolve** method. The collection returned in **AmbiguousNames** represents the [AddressEntry](#) objects in the address book that could resolve to the supplied recipient name.

If the collection returned in the **AmbiguousNames** property is empty, there are no candidates for the supplied recipient name, and it should be considered unresolvable. If the collection contains address entries, they can be displayed to the user so that the appropriate one can be selected. The **GetFirstUnresolved** and **GetNextUnresolved** methods can be used to find all the ambiguous recipients in a [Recipients](#) collection.

Although the **AmbiguousNames** property itself is read-only, the collection it returns can be accessed in the normal manner through its **Add** and **Delete** methods, and the properties on its member [AddressEntry](#) objects retain their respective read/write or read-only accessibility.

Example

```
' function to attempt ambiguous name resolution (ANR)
Function TryANR(objRecip As Recipient) As Boolean
Dim objAmbigEntries As AddressEntries ' possible resolutions
Dim strChosenID As String ' ID of address entry chosen by user
' ... set up error handling ...
Set objAmbigEntries = objRecip.AmbiguousNames
If objAmbigEntries Is Nothing Then
    MsgBox "No eligible names for resolution"
    objRecip.Delete ' nothing else can be done at this point
    TryANR = False
Else
    ' show address entries to user so one can be chosen, and save its
    ' entry identifier: strChosenID = objAddrEntry.ID
    objRecip.ID = strChosenID
    TryANR = True
End If
End Function
```

Delete Method (Recipient Object)

Group

The **Delete** method removes the Recipient object from the Recipients collection.

Syntax

objRecipient.Delete()

Remarks

The **Delete** method performs an irreversible operation on the collection. It calls **Release** on the collection's reference to the Recipient object. If you have another reference to the recipient, you can still access its properties and methods, but you can never again associate it with any collection because the **Add** method always creates a new object. You should **Set** your reference variable either to **Nothing** or to another recipient.

The final **Release** on the Recipient object takes place when you assign your reference variable to **Nothing**, or when you call **Delete** if you had no other reference. At this point the object is removed from memory. Attempted access to a released object results in an error return of **CdoE_INVALID_OBJECT**.

When you delete a member of a collection, the collection is immediately refreshed, meaning that its **Count** property is reduced by one and its members are reindexed. To access a member following the deleted member, you must use its new index value. For more information, see Looping Through a Collection.

The effect of the **Delete** operation is not permanent until you use the **Update**, **Send**, or **Delete** method on the Message object to which this recipient belongs.

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.

Example

This code fragment illustrates the two situations previously explained. The **Set** statement calls **AddRef** on the first Recipient object. That reference survives the call to **Delete** and has to be reassigned. The second Recipient object is deleted without creating another reference, and no other action is necessary.

```
' assume valid Message object
Set objRecipient = objMessage.Recipients.Item(1)
objRecipient.Delete ' still have a reference from Set statement
' ... other operations on objRecipient possible but pointless ...
Set objRecipient = Nothing ' necessary to remove reference
' ...
objMessage.Recipients.Item(2).Delete ' no reference to remove
```

DisplayType Property (Recipient Object)

Group

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

Syntax

objRecipient.**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 recipients.

The following values are defined:

| DisplayType value | Decimal value | Description |
|---------------------------|----------------------|---|
| CdoAgent | 3 | An automated agent, such as Quote-of-the-Day. |
| CdoDistList | 1 | A public distribution list. |
| CdoForum | 2 | A forum, such as a bulletin board or a public folder. |
| CdoOrganization | 4 | A special address entry defined for large groups, such as a helpdesk. |
| CdoPrivateDistList | 5 | A private, personally administered distribution list. |
| CdoRemoteUser | 6 | A messaging user in a remote messaging system. |
| CdoUser | 0 | A local messaging user. |

When you **Add** a new recipient to a [Recipients](#) collection, the **DisplayType** property is set by the address book provider to either **CdoUser** or **CdoDistList**, depending on which kind of recipient is being added. The **DisplayType** property cannot subsequently be changed.

A private distribution list (PDL) exists only in your personal address book (PAB) and does not have an e-mail address. Before invoking a recipient's [Address](#) property, you should verify that its **DisplayType** is not **CdoPrivateDistList**. Attempted access to addressing properties on a PDL results in a return of **CdoE_NOT_FOUND**.

GetFreeBusy Method (Recipient Object)

Group

The **GetFreeBusy** method returns a string representing the availability of the recipient for a meeting over a specified period of time.

Syntax

strAvail = *objRecipient*.**GetFreeBusy**(*StartTime*, *EndTime*, *Interval*)

strAvail

On successful return, contains a string indicating the recipient's availability for each of the time slots in the specified time period.

objRecipient

Required. The Recipient object.

StartTime

Required. Variant (**vbDate** format). Specifies the date/time of the beginning of the first time slot.

EndTime

Required. Variant (**vbDate** format). Specifies the date/time of the end of the last time slot.

Interval

Required. Long. Specifies the length of each time slot in minutes. If this parameter is less than 1, **GetFreeBusy** returns **CdoE_INVALID_PARAMETER**.

Remarks

The returned string length equals the number of time slots between *StartTime* and *EndTime*. Each character is the ASCII representation of the appropriate type library constant indicating the recipient's availability during a time slot:

| ASCII character | Corresponding type library constant | Meaning |
|-----------------|-------------------------------------|--|
| "0" | CdoFree | Available for appointments or meetings throughout the time slot |
| "1" | CdoTentative | At least one tentative commitment during the time slot |
| "2" | CdoBusy | At least one confirmed commitment during the time slot |
| "3" | CdoOutOfOffice | Designated as out-of-office (OOO) for at least part of the time slot |

If there is any overlapping of commitments during a time slot, **GetFreeBusy** returns the most committed state, that is, the highest character value. For example, if a recipient already has one tentative meeting and one confirmed meeting scheduled during the same time slot, **GetFreeBusy** returns "2" for that time slot, corresponding to **CdoBusy**. **CdoFree** is not returned unless the entire time slot is free of commitments.

For performance reasons, calendaring clients typically do not publish appointments indefinitely into the future. For example, the CDO Library, Microsoft® Outlook™, Microsoft® Schedule+, and Microsoft Outlook Web Access (OWA) all publish appointments for a default maximum of three months past the current date. This means that if you call **GetFreeBusy** on a messaging user and specify time slots more than three months in advance of that user's current date, **GetFreeBusy** is likely to return **CdoFree** for every such time slot. Unless you are familiar with the behavior and publishing limits of a messaging user's calendaring client, you should not interpret **CdoFree** as meaning that the user is

genuinely free for a time slot significantly in the future.

When a messaging user creates an appointment using Outlook, the calendar is not updated until logoff from Outlook. This means that changes to that user's free/busy information are not available to other messaging users until the Outlook user logs off.

When a messaging user's mailbox is first created using either Outlook or a CDO application, its free/busy information is not initialized until the user calls a method, such as **GetDefaultFolder** or **GetFreeBusy**, that initializes this information. If another messaging user calls **GetFreeBusy** on that user before the free/busy information is initialized, CDO returns **CdoE_NOT_FOUND**. This is not the case with a Schedule+ mailbox, which initializes its free/busy information immediately upon creation.

If a recipient represents a distribution list, the status of its individual members cannot be returned to you. A meeting request should be sent only to single messaging users. You can determine if a messaging user is a distribution list by checking the **DisplayType** property of the **AddressEntry** object representing that user. You can obtain the **AddressEntry** object underlying a recipient from the Recipient object's **AddressEntry** property.

ID Property (Recipient Object)

Group

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. However, MAPI does not require identifier values to be binary comparable. Accordingly, two identifier values can be different, yet refer to the same object. MAPI compares identifiers with the **CompareEntryIDs** method. CDO provides the **CompareIDs** method in the [Session](#) object. For more information on entry identifiers, see the *MAPI Programmer's Reference*.

Although the [AddressEntry](#) and Recipient objects are not identical objects in the CDO Library, they represent the same underlying MAPI messaging user object, and the address entry's **ID** property is equal to the recipient's **ID** property. This can be used to advantage, for example, when adding an existing AddressEntry object to a [Recipients](#) collection. You can use the address entry's **ID** property as the *entryID* parameter to the **Add** method.

If you attempt to set the **ID** property to an empty string, **CdoE_INVALID_PARAMETER** is returned.

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

Index Property (Recipient Object)

Group

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 be saved and used later with the collection's **Item** property to reselect the same recipient in the collection.

The first object in the collection has an **Index** value of 1.

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

```
Dim curIndex, savIndex as Integer ' variables to work with Index
' select next recipient from collection
If curIndex >= objRecipColl.Count Then
    curIndex = objRecipColl.Count
    MsgBox "Already at end of recipient list"
    Exit Function
End If
' index is < count; can be incremented by 1
curIndex = curIndex + 1
Set objOneRecip = objRecipColl.Item(curIndex)
' could be objRecipColl(curIndex) since Item is default property
' save index for later use; but remember it could change if deletions
If objOneRecip Is Nothing Then
    MsgBox "Could not select next recipient"
    Exit Function
End If
savIndex = objOneRecip.Index
MsgBox "Recipient index = " & savIndex
```

IsSameAs Method (Recipient Object)

Group

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

Syntax

boolSame = *objRecipient*.**IsSameAs**(*objRecip2*)

boolSame

On successful return, contains **True** if the two objects are the same.

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 in the underlying messaging system. 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. This is necessary because, although MAPI requires all entry identifiers to be unique, it does not require two of them identifying the same object to be identical. A generic comparison of any two objects' unique identifiers is also available with the Session object's **CompareIDs** method.

MeetingResponseStatus Property (Recipient Object) Group

The **MeetingResponseStatus** property returns or sets the status of this recipient's response to a meeting request. Read/write.

Syntax

objRecipient.**MeetingResponseStatus**

Data Type

Long

Remarks

A messaging user that has created an [AppointmentItem](#) object can turn it into a meeting by setting its **MeetingStatus** property to **CdoMeeting** and sending it to one or more recipients. That user can then monitor the responses with the **MeetingResponseStatus** property. The original appointment underlies the meeting and retains its [Recipients](#) collection, and the originating user can read the **MeetingResponseStatus** property of each Recipient in the collection to determine the current status of that recipient's response.

An [AddressEntry](#) object representing the originating messaging user is available through the appointment's [Organizer](#) property.

The **MeetingResponseStatus** property can have exactly one of the following values:

| MeetingResponseStatus value | Decimal value | Description |
|------------------------------------|----------------------|--|
| CdoResponseNone | 0 | Used by Microsoft® Outlook™ to indicate that this recipient has not responded to the meeting request. |
| CdoResponseNotResponded | 5 | This recipient has not responded to the meeting request. |
| CdoResponseAccepted | 3 | This recipient has responded to the meeting request with a firm acceptance. |
| CdoResponseDeclined | 4 | This recipient has responded to the meeting request with a declination. |
| CdoResponseTentative | 2 | This recipient has responded to the meeting request with a tentative acceptance. |
| CdoResponseOrganized | 1 | Used by Microsoft® Schedule+ to indicate that this recipient has not responded to the meeting request. |

You can get the meeting status for the underlying [AppointmentItem](#) object from its **MeetingResponseStatus** property.

Name Property (Recipient Object)

Group

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

Syntax

objRecipient.Name

The **Name** property is the default property of a Recipient object, meaning that *objRecipient* is syntactically equivalent to *objRecipient*.Name in Microsoft® Visual Basic® code.

Data Type

String

Remarks

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

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)

Group

The **Resolve** method resolves a recipient's address information into a full messaging address.

Syntax

objRecipient.**Resolve**([*showDialog*])

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 have supplied the Recipient object'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.

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.

The **Resolve** method validates the Recipient object's **Type** property and returns **CdoE_INVALID_PARAMETER** if it is not one of the defined recipient types.

You can call the **Recipients** collection's **Resolve** method to resolve every object in the collection and also force an update to the collection's **Count** property.

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 book or books specified in the profile, such as the global address list (GAL) and 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 dialog boxes:

- **Details** method (**AddressEntry** object)
- **Options** and **Send** methods (**Message** object)
- **Resolve** method (Recipient object)
- **Resolve** method (**Recipients** collection)
- **AddressBook** and **Logon** methods (**Session** object)

However, if your application is running as a Microsoft® Windows NT® service, for example from Active Server Pages (ASP) script on a Microsoft® Internet Information Server (IIS), no user interface is allowed.

For more information on Windows NT services, see the Win32® Web page *Using MAPI from a Windows NT Service* at <http://www.microsoft.com/win32dev/mapi/mapiserv.htm>. For more information on running as a service, see "Windows NT Service Client Applications" in the *MAPI Programmer's*

Reference.

Type Property (Recipient Object)

Group

The **Type** property specifies the recipient type of the Recipient object, that is, whether it is a To, Cc, or Bcc recipient. Read/write.

Syntax

objRecipient.Type

Data Type

Long

Remarks

The **Type** property has the following defined values:

| Recipient type | Value | Description |
|----------------|-------|--|
| CdoTo | 1 | The recipient is on the To line (default). |
| CdoCc | 2 | The recipient is on the Cc line. |
| CdoBcc | 3 | The recipient is on the Bcc line. |

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

See Also

[Address Property \(Recipient Object\)](#)

Recipients Collection Object

The Recipients collection object contains one or more Recipient objects and specifies the recipients of a message.

At a Glance

| | |
|----------------------------|---------------------------|
| Specified in type library: | CDO.DLL |
| First available in: | CDO 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 Microsoft® Visual Basic® **For Each** statement. For more information on collections, see [Object Collections](#).

Properties

| Name | Available since 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 |
| RawTable | 1.1 | IUnknown object | Read/write
(Note: Not available to Visual Basic applications) |
| 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 since version | Parameters |
|------------------------------------|-------------------------|--|
| Add | 1.0.a | (optional) <i>name</i> as String ,
(optional) <i>address</i> as String ,
(optional) <i>type</i> as Long ,
(optional) <i>entryID</i> as String |
| AddMultiple | 1.1 | <i>names</i> as String ,
(optional) <i>type</i> as Long |
| Delete | 1.0.a | (none) |
| GetFirstUnresolved | 1.1 | (none) |

| | | |
|--------------------------|-------|--|
| <u>GetFreeBusy</u> | 1.2 | <i>StartTime</i> as Variant ,
<i>EndTime</i> as Variant ,
<i>Interval</i> as Long |
| <u>GetNextUnresolved</u> | 1.1 | (none) |
| <u>Resolve</u> | 1.0.a | (optional) <i>showDialog</i> as Boolean |

Remarks

A Recipients collection can be rendered into HTML hypertext in tabular form using the CDO Rendering ContainerRenderer object. To specify this, set the container renderer's **DataSource** property to the Recipients collection object itself.

With the same **DataSource** setting, the container renderer's RenderProperty method can also render selected properties of the collection's parent Message object. The individual properties that can be rendered are indicated in the Message object property descriptions.

Add Method (Recipients Collection)

Group

The **Add** method creates and returns a new Recipient object in the Recipients collection.

Syntax

Set *objRecipient* = *objRecipColl*.**Add**([*name*] [, *address*] [, *type*] [, *entryID*])

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. The *name* parameter is ignored if the *entryID* parameter is supplied.

address

Optional. String. The full messaging address of the recipient. When this parameter is not present, the new Recipient object's **Address** property is set to an empty string. The *address* parameter is ignored if the *entryID* parameter is supplied.

type

Optional. Long. The recipient type; the initial value for the new recipient's **Type** property. The following values are valid:

| Recipient type | Value | Description |
|----------------|-------|--|
| CdoTo | 1 | The recipient is on the To line (default). |
| CdoCc | 2 | The recipient is on the Cc line. |
| CdoBcc | 3 | The recipient is on the Bcc line. |

The *type* parameter applies whether the *entryID* parameter is furnished or not.

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 *name* and *address* parameters are not used. When it is not present, the method uses the *name*, *address*, and *type* parameters to define 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 *name* and *address* parameters are not used.

The *address* parameter, if set, must contain a *full address*, such as that contained in the recipient's **Address** property. An AddressEntry object's **Address** property is not a full address because it does not contain the address type information found in the AddressEntry object's **Type** property. If the user you are adding is represented by an AddressEntry object, such as is returned by the Session object's **CurrentUser** property, you must concatenate its **Type** and **Address** properties with a connecting colon to construct the full address.

When no parameters are present, an empty Recipient object is created.

The **DisplayType** property of the new Recipient object is set by the address book provider to either **CdoUser** or **CdoDistList**, depending on which kind of recipient is being added. The **DisplayType** property is read-only and cannot subsequently be changed.

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 new 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 is not needed for 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:=CdoTo)
If objOneRecip Is Nothing Then
    MsgBox "Unable to add recipient using name and type"
    Exit Function
End If
objOneRecip.Resolve ' find its full address in address book

' 2. add a custom recipient
Set objOneRecip = objNewMessage.Recipients.Add( _
                                               Address:="SMTP:johndoe@company.com", _
                                               Type:=CdoTo)
If objOneRecip Is Nothing Then
    MsgBox "Unable to add recipient using custom address"
    Exit Function
End If
objOneRecip.Resolve ' make it an object and give it an entry ID

' 3. add a valid address entry object, such as Message.Sender
' assume valid address entry ID from an existing message
Set objOneRecip = objNewMessage.Recipients.Add( _
                                               entryID:=strAddrEntryID)
'     or .Add( , , , strAddrEntryID) if you can't use named parameters
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)
```

AddMultiple Method (Recipients Collection) Group

The **AddMultiple** method creates zero or more new [Recipient](#) objects in the Recipients collection.

Syntax

objRecipColl.AddMultiple(*names* [, *type*])

objRecipColl

Required. The Recipients collection object.

names

Required. String. A list of zero or more resolvable recipient strings delimited by semicolons. Each resolvable string can be a messaging user's display name or a messaging address with or without the address type. A messaging address without address type must be an SMTP (Simple Mail Transfer Protocol) address usable on the Internet.

type

Optional. Long. The recipient type; the initial value for the **Type** property that is to apply to all the new recipients. The following values are valid:

| Recipient type | Value | Description |
|----------------|-------|--|
| CdoTo | 1 | The recipients are on the To line (default). |
| CdoCc | 2 | The recipients are on the Cc line. |
| CdoBcc | 3 | The recipients are on the Bcc line. |

Remarks

The **AddMultiple** method is useful when responding to standard e-mail forms that invite the user to enter a series of recipients in a connected list, such as the To and Cc lines for a Microsoft® Exchange Client message.

The **AddMultiple** method does not resolve the new recipients. You must call either each recipient's **Resolve** method or the Recipients collection's **Resolve** method following the **AddMultiple** call.

Example

This code fragment illustrates the different possibilities of resolvable recipient strings:

```
Dim name1, name2, name3, toStr As String
name1 = "John Doe" ' display name
name2 = "jimdoe@company.com" ' SMTP address without address type
name3 = "Jane Doe[SMTP:janedoe@company.com]" ' full messaging address
toStr = name1 & ";" & name2 & ";" & name3
objRecipients.AddMultiple (toStr, CdoTo)
objRecipients.Resolve()
```

Count Property (Recipients Collection)

Group

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 message's Recipients collection to another message's collection. It shows the **Count** and **Item** properties working together. Note how much more code this requires than copying the Message object's **Recipients** property from the original message to the copy.

```
' from the sample function Util_CopyMessage
' Copy all Recipient objects from one message's collection to another
Dim objOneMsg, objCopyMsg as Message
Dim objRecipColl as Recipients ' source message Recipients collection
Dim objOneRecip as Recipient ' individual recipient in target message
' ... verify valid messages ...
Set objRecipColl = objOneMsg.Recipients
For i = 1 To objRecipColl.Count Step 1
    strRecipName = objRecipColl.Item(i).Name
    ' could be objRecipColl(i).Name since Item is default property
    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 = objRecipColl.Item(i).Address
        objOneRecip.Type = objRecipColl.Item(i).Type
    End If
Next i
```

Delete Method (Recipients Collection)

Group

The **Delete** method removes all the Recipient objects from the Recipients collection.

Syntax

objRecipColl.Delete()

Remarks

The **Delete** method performs an irreversible operation on the collection. It calls **Release** on the collection's reference to every Recipient object. If you have another reference to a recipient, you can still access its properties and methods, but you can never again associate it with any collection because the **Add** method always creates a new object. You should **Set** your reference variable either to **Nothing** or to another recipient.

The final **Release** on each Recipient object takes place when you assign your reference variable to **Nothing**, or when you call **Delete** if you had no other reference. At this point the object is removed from memory. Attempted access to a released object results in an error return of **CdoE_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. A permanently deleted member cannot be recovered. However, the collection itself is still valid, and you can **Add** new members to it.

GetFirstUnresolved Method (Recipients Collection) Group

The **GetFirstUnresolved** method returns the first unresolved Recipient object in the Recipients collection. It returns **Nothing** if there are no unresolved recipients in the collection.

Syntax

Set *objRecipient* = *objRecipColl*.**GetFirstUnresolved**()

objRecipient

On successful return, represents the first unresolved recipient in the collection.

objRecipColl

Required. The Recipients collection object.

Remarks

The **GetFirstUnresolved** and **GetNextUnresolved** methods can be used to find all the ambiguous recipient names in a Recipients collection.

GetFreeBusy Method (Recipients Collection) Group

The **GetFreeBusy** method returns a string representing the combined availability of all recipients for a meeting over a specified period of time.

Syntax

strAvail = *objRecipColl*.**GetFreeBusy**(*StartTime*, *EndTime*, *Interval*)

strAvail

On successful return, contains a string indicating the recipients' availability for each of the time slots in the specified time period.

objRecipColl

Required. The Recipients collection object.

StartTime

Required. Variant (**vbDate** format). Specifies the date/time of the beginning of the first time slot.

EndTime

Required. Variant (**vbDate** format). Specifies the date/time of the end of the last time slot.

Interval

Required. Long. Specifies the length of each time slot in minutes. If this parameter is less than 1, **GetFreeBusy** returns **CdoE_INVALID_PARAMETER**.

Remarks

The returned string length equals the number of time slots between *StartTime* and *EndTime*. Each character is the ASCII representation of the appropriate type library constant indicating the recipients' combined availability during a time slot:

| ASCII character | Corresponding type library constant | Meaning |
|-----------------|-------------------------------------|--|
| "0" | CdoFree | Available for appointments or meetings throughout the time slot |
| "1" | CdoTentative | At least one tentative commitment during the time slot |
| "2" | CdoBusy | At least one confirmed commitment during the time slot |
| "3" | CdoOutOfOffice | Designated as out-of-office (OOO) for at least part of the time slot |

If there is any overlapping of commitments during a time slot, **GetFreeBusy** returns the most committed state, that is, the highest character value. For example, if one recipient already has a tentative meeting and another has a confirmed meeting scheduled during the same time slot, **GetFreeBusy** returns "2" for that time slot, corresponding to **CdoBusy**. **CdoFree** is not returned unless the entire time slot is free of commitments.

For performance reasons, calendaring clients typically do not publish appointments indefinitely into the future. For example, the CDO Library, Microsoft® Outlook™, Microsoft® Schedule+, and Microsoft Outlook Web Access (OWA) all publish appointments for a default maximum of three months past the current date. This means that if you call **GetFreeBusy** on a messaging user and specify time slots more than three months in advance of that user's current date, **GetFreeBusy** is likely to return **CdoFree** for every such time slot. Unless you are familiar with the behavior and publishing limits of a messaging user's calendaring client, you should not interpret **CdoFree** as meaning that the user is

genuinely free for a time slot significantly in the future.

When a messaging user creates an appointment using Outlook, the calendar is not updated until logoff from Outlook. This means that changes to that user's free/busy information are not available to other messaging users until the Outlook user logs off.

When a messaging user's mailbox is first created using either Outlook or a CDO application, its free/busy information is not initialized until the user calls a method, such as **GetDefaultFolder** or **GetFreeBusy**, that initializes this information. If another messaging user calls **GetFreeBusy** on that user before the free/busy information is initialized, CDO returns **CdoE_NOT_FOUND**. This is not the case with a Schedule+ mailbox, which initializes its free/busy information immediately upon creation.

If a recipient represents a distribution list, the status of its individual members cannot be returned to you. A meeting request should be sent only to single messaging users. You can determine if a messaging user is a distribution list by checking the **DisplayType** property of the **AddressEntry** object representing that user. You can obtain the **AddressEntry** object underlying a recipient from that **Recipient** object's **AddressEntry** property.

GetNextUnresolved Method (Recipients Collection) Group

The **GetNextUnresolved** method returns the next unresolved Recipient object in the Recipients collection. It returns **Nothing** if there are no unresolved recipients remaining in the collection.

Syntax

Set *objRecipient* = *objRecipColl*.**GetNextUnresolved**()

objRecipient

On successful return, represents the next unresolved recipient in the collection.

objRecipColl

Required. The Recipients collection object.

Remarks

The **GetFirstUnresolved** and **GetNextUnresolved** methods can be used to find all the ambiguous recipient names in a Recipients collection.

Item Property (Recipients Collection)

Group

The **Item** property returns a single Recipient object from the Recipients collection. Read-only.

Syntax

objRecipColl.**Item**(*index*)

index

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

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

Data Type

Object (Recipient)

Remarks

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

Although the **Item** property itself is read-only, the Recipient object it returns can be accessed in the normal manner, and its properties retain their respective read/write or read-only accessibility.

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
```

RawTable Property (Recipients Collection) Group

The **RawTable** property returns an **IUnknown** pointer to the MAPI table object underlying the Recipients collection. Not available to Microsoft® Visual Basic® applications. Read/write.

Syntax

objRecipColl.RawTable

Data Type

Variant (**vbDataObject** format)

Remarks

The **RawTable** property is not available to Visual Basic programs. It is accessible only by C/C++ programs that deal with **IUnknown** objects. Visual Basic supports the **IDispatch** interface and not **IUnknown**. The **RawTable** property is an **IUnknown** object that returns an **IMAPITable** interface in response to **QueryInterface**. For more information, see [Introduction to Automation](#) and [How Programmable Objects Work](#). Also see the "COM and ActiveX Object Services" section of the Microsoft Platform SDK.

If your application uses any **MAPIOBJECT** or **RawTable** properties, it must **Release** them all before calling the [Session](#) object's **Logoff** method. Failure to do so can result in unexpected behavior.

The **RawTable** property does not correspond to a MAPI property and cannot be rendered into HTML hypertext by the CDO Rendering Library.

Resolve Method (Recipients Collection)

Group

The **Resolve** method traverses the Recipients collection to resolve every recipient's address information into a full messaging address.

Syntax

objRecipColl.**Resolve**([*showDialog*])

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** property 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 dialog boxes:

- **Details** method (AddressEntry object)
- **Options** and **Send** methods (Message object)
- **Resolve** method (Recipient object)
- **Resolve** method (Recipients collection)
- **AddressBook** and **Logon** methods (Session object)

However, if your application is running as a Microsoft® Windows NT® service, for example from Active Server Pages (ASP) script on a Microsoft® Internet Information Server (IIS), no user interface is allowed.

For more information on Windows NT services, see the Win32® Web page *Using MAPI from a Windows NT Service* at <http://www.microsoft.com/win32dev/mapi/mapiserv.htm>. For more information on running as a service, see "Windows NT Service Client Applications" in the *MAPI Programmer's Reference*.

Example

```
' from the sample function Util_NewConversation
' create a valid new message object in the Outbox
  With objNewMsg
    .Subject = "used car wanted"
    ' ... set other properties here ...
    Set objOneRecip = .Recipients.Add(Name:="Car Ads", _
                                  Type:=CdoTo)

    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.

RecurrencePattern Object

The RecurrencePattern object describes the recurrence pattern for an AppointmentItem object.

At a Glance

| | |
|----------------------------|---------------------------------|
| Specified in type library: | CDO.DLL |
| First available in: | CDO Library version 1.2 |
| Parent objects: | AppointmentItem |
| Child objects: | (none) |
| Default property: | (none) |

Properties

| Name | Available since version | Type | Access |
|----------------------------------|-------------------------|---------------------------------|------------|
| Application | 1.2 | String | Read-only |
| Class | 1.2 | Long | Read-only |
| DayOfMonth | 1.2 | Long | Read/write |
| DayOfWeekMask | 1.2 | Long | Read/write |
| Duration | 1.2 | Long | Read-only |
| EndTime | 1.2 | Variant (vbDate format) | Read/write |
| Instance | 1.2 | Long | Read/write |
| Interval | 1.2 | Long | Read/write |
| MonthOfYear | 1.2 | Long | Read/write |
| NoEndDate | 1.2 | Boolean | Read/write |
| Occurrences | 1.2 | Long | Read/write |
| Parent | 1.2 | AppointmentItem object | Read-only |
| PatternEndDate | 1.2 | Variant (vbDate format) | Read/write |
| PatternStartDate | 1.2 | Variant (vbDate format) | Read/write |
| RecurrenceType | 1.2 | Long | Read/write |
| Session | 1.2 | Session object | Read-only |
| StartTime | 1.2 | Variant (vbDate format) | Read/write |

Methods

(None.)

Remarks

A RecurrencePattern object contains properties that fully specify the recurrence characteristics of an [AppointmentItem](#) object. It is created and linked to the appointment when you first call the AppointmentItem object's [GetRecurrencePattern](#) method. This sets the appointment's [IsRecurring](#) property to **True** and instantiates a new RecurrencePattern object populated with the default values

indicated in the property descriptions.

The `RecurrencePattern` object applies only to its parent appointment and cannot be used for any other `AppointmentItem` object. The recurrence pattern can be accessed from any appointment in the recurring series, that is, from an individual recurrence as well as from the appointment that originated the series. The originating appointment can be obtained with the `RecurrencePattern` object's **Parent** property.

To edit the entire recurring series of appointments, you modify the appropriate properties on either the originating `AppointmentItem` object or its child `RecurrencePattern` object. To edit an individual recurrence only, you instantiate it and modify its `AppointmentItem` properties. You can instantiate an individual recurrence by using a `MessageFilter` object to restrict the `Messages` collection containing the appointments. All changes take effect when you call the appointment's **Send** or **Update** method.

If any change is made to the recurrence pattern after one or more individual recurrences have been instantiated, some or all of them may be automatically deleted. Microsoft® Outlook™ deletes all individual recurrences, while Microsoft® Schedule+ deletes only those that are earlier than the new **PatternStartDate** or later than the new **PatternEndDate**. CDO determines what your active calendar store is and deletes recurrences in the appropriate manner.

If you instantiate one or more individual recurrences and subsequently delete them all, Microsoft Outlook deletes the `RecurrencePattern` object. If CDO determines that your active calendar store is Outlook, it also deletes the `RecurrencePattern` object in this case.

The `AppointmentItem` object's **ClearRecurrencePattern** method resets **IsRecurring** to **False** and calls **Release** on the `RecurrencePattern` object. This is normally the final **Release** because the `RecurrencePattern` object should have no other references. The `RecurrencePattern` object is removed from memory in response to its final **Release**.

The **RecurrenceType** property determines the *recurrence unit*, the basic time unit for recurrence of the appointment. This can be a day, a week, a month, or a year. The appointment can recur on every instance of this recurrence unit, on isolated instances selected by the **Instance** property, or on periodic instances defined by the **Interval** property.

The **DayOfMonth**, **DayOfWeekMask**, and **MonthOfYear** properties specify the days and months when the appointment is to recur. The **StartTime** and **EndTime** properties determine the times of day for each occurrence. The **PatternStartDate**, **PatternEndDate**, **Occurrences**, and **NoEndDate** properties define the overall time period during which the appointment is to recur.

Several of the recurrence pattern properties have interdependent values. When you set one of these properties, related properties are forced into conformance in order to ensure consistency. For example, changing **PatternEndDate** causes **Occurrences** to be recalculated, and changing **Occurrences** causes **PatternEndDate** to be recalculated. The most recent change determines the settings of the interdependent properties. Each property description includes the effects of changing its value.

The pattern you specify in the **DayOfMonth**, **DayOfWeekMask**, **Instance**, **Interval**, and **MonthOfYear** properties is not required to include a recurrence on the day of the original appointment, nor on the days indicated in **PatternStartDate** or **PatternEndDate**. These days are counted in **Occurrences** only if they match the pattern, and a recurrence is generated on the starting or ending day only if that day matches the pattern.

This code fragment defines recurrence patterns for three `AppointmentItem` objects obtained from the `Messages` collection of a calendar folder. The first specifies recurrence on the third Friday of every month. The second and third specify recurrence on American Thanksgiving and Canadian Thanksgiving respectively.

```
Dim objApp3rdFri, objAppAmThx, objAppCanThx As AppointmentItem
Dim objRec3rdFri, objRecAmThx, objRecCanThx As RecurrencePattern
' ... assume all three AppointmentItem objects are valid ...
' ... calling the GetRecurrencePattern method makes them recurring ...
```

```
Set objRec3rdFri = objApp3rdFri.GetRecurrencePattern
Set objRecAmThx = objAppAmThx.GetRecurrencePattern
Set objRecCanThx = objAppCanThx.GetRecurrencePattern
' every third Friday
objRec3rdFri.RecurrenceType = CdoRecurTypeMonthlyNth
objRec3rdFri.DayOfWeekMask = CdoFriday
objRec3rdFri.Instance = 3 ' third instance of selected day
objApp3rdFri.Update ' needed for settings to take effect in calendar
' American Thanksgiving (fourth Thursday of November)
objRecAmThx.RecurrenceType = CdoRecurTypeYearlyNth
objRecAmThx.DayOfWeekMask = CdoThursday
objRecAmThx.MonthOfYear = 11 ' November
objRecAmThx.Instance = 4
objAppAmThx.Update ' needed for settings to take effect in calendar
' Canadian Thanksgiving (second Monday of October)
objRecCanThx.RecurrenceType = CdoRecurTypeYearlyNth
objRecCanThx.DayOfWeekMask = CdoMonday
objRecCanThx.MonthOfYear = 10 ' October
objRecCanThx.Instance = 2
objAppCanThx.Update ' needed for settings to take effect in calendar
```


DayOfMonth Property (RecurrencePattern Object)

Group

The **DayOfMonth** property returns or sets the day of the month on which the appointment recurs. Read/write.

Syntax

objRecurPatt.DayOfMonth

Data Type

Long

Remarks

The **DayOfMonth** property contains the calendar date of each month on which the [AppointmentItem](#) is to recur, for example the value 1 to indicate the first day of every month. The last day of every month can be represented by the value 31 in monthly recurrences. For yearly recurrences, **DayOfMonth** cannot be set past the last possible day in the selected month.

If an appointment is viewed through Microsoft® Schedule+ and its **DayOfMonth** property has a value greater than the number of days in the specified month, Schedule+ does not generate a recurrence for that month. This applies, for example, if **DayOfMonth** is 29 and **MonthOfYear** is 2. For such an appointment, Schedule+ only generates a recurrence in leap years, whereas Microsoft® Outlook™ and CDO generate recurrences on February 28 for years other than leap years.

DayOfMonth is only valid if the value of the **RecurrenceType** property is **CdoRecurTypeMonthly** or **CdoRecurTypeYearly**. When **DayOfMonth** is valid on a newly created RecurrencePattern object, it defaults to the current day of the month.

Note If the [AppointmentItem](#) object has been made into a meeting, the **DayOfMonth**, **DayOfWeekMask**, and **MonthOfYear** properties are all held internally in the meeting organizer's current time zone. If these properties are displayed or read by a messaging user in a different time zone, they are not converted. The user or application accessing these properties may need to be aware that they represent the organizer's time zone.

Changes you make to properties on a RecurrencePattern object take effect when you call the underlying appointment's **Send** or **Update** method.

DayOfWeekMask Property (RecurrencePattern Object)

Group

The **DayOfWeekMask** property returns or sets the mask for the days of the week on which the appointment recurs. Read/write.

Syntax

objRecurPatt.DayOfWeekMask

Data Type

Long

Remarks

The **DayOfWeekMask** property contains the days of the week on each of which the [AppointmentItem](#) is to recur. It can include the following values in any combination:

| DayOfWeekMask setting | Decimal value | Meaning |
|-----------------------|---------------|---------------------------------------|
| CdoSunday | 1 | The appointment recurs on Sundays. |
| CdoMonday | 2 | The appointment recurs on Mondays. |
| CdoTuesday | 4 | The appointment recurs on Tuesdays. |
| CdoWednesday | 8 | The appointment recurs on Wednesdays. |
| CdoThursday | 16 | The appointment recurs on Thursdays. |
| CdoFriday | 32 | The appointment recurs on Fridays. |
| CdoSaturday | 64 | The appointment recurs on Saturdays. |

The maximum value for the **DayOfWeekMask** property is 127, which is the logical inclusive **OR** of all seven days. An attempt to set **DayOfWeekMask** to any value less than 1 or greater than 127 results in a return of **CdoE_INVALID_PARAMETER**.

Note that the **DayOfWeekMask** property is not compatible with the CDO Rendering [ContainerRenderer](#) object's **FirstDayOfWeek** property or the [Session](#) object's "FirstDayOfWeek" option, which use an enumeration starting with 1 for Monday and ending with 7 for Sunday. The session options are obtained with the **GetOption** method and set with the **SetOption** method. The Session object's "WorkingDays" option, however, is compatible with the mask constants used by **DayOfWeekMask**.

DayOfWeekMask is only valid if the value of the **RecurrenceType** property is **CdoRecurTypeWeekly**, **CdoRecurTypeMonthlyNth**, or **CdoRecurTypeYearlyNth**. When **DayOfWeekMask** is valid on a newly created RecurrencePattern object, it defaults to the current day of the week.

Setting **DayOfWeekMask** to multiple days per week is only valid if the value of the **RecurrenceType** property is **CdoRecurTypeWeekly**. Recurrences of type **CdoRecurTypeMonthlyNth** or **CdoRecurTypeYearlyNth** can only use a single day per week.

Note If the [AppointmentItem](#) object has been made into a meeting, the **DayOfMonth**, **DayOfWeekMask**, and **MonthOfYear** properties are all held internally in the meeting organizer's current time zone. If these properties are displayed or read by a messaging user in a different time zone, they are not converted. The user or application accessing these properties may need to be aware that they represent the organizer's time zone.

Changes you make to properties on a RecurrencePattern object take effect when you call the

underlying appointment's Send or Update method.

Duration Property (RecurrencePattern Object) Group

The **Duration** property returns the duration of the recurring appointment in minutes. Read-only.

Syntax

objRecurPatt.Duration

Data Type

Long

Remarks

The **Duration** property contains the number of minutes the appointment is to last every time it recurs. **Duration** is always valid on a newly created RecurrencePattern object and defaults to the value of the **Duration** property of the AppointmentItem object that created this recurrence pattern.

The minimum value of **Duration** is 0. The maximum value allows an appointment to last until its next possible recurrence, and is dependent on the settings of the RecurrenceType, Interval, and DayOfWeekMask properties, as follows:

| RecurrenceType setting | Maximum Duration value |
|--|---|
| CdoRecurTypeDaily | (1 day) x (Interval value) |
| CdoRecurTypeMonthly | (28 days) x (Interval value) |
| CdoRecurTypeMonthlyNth | |
| CdoRecurTypeYearly | 336 days, that is, 48 weeks, since |
| CdoRecurTypeYearlyNth | Interval cannot be greater than 1 for yearly recurrence types |
| CdoRecurTypeWeekly ,
and DayOfWeekMask specifies exactly one day per week | (7 days) x (Interval value) |
| CdoRecurTypeWeekly ,
and DayOfWeekMask specifies more than one day per week | the minimum difference between any two days specified in the mask, for example 3 days if the mask specifies CdoMonday and CdoFriday |

Among its possible values, the **Duration** property can be any multiple of 24 hours that does not exceed its maximum value. If an occurrence has such a value for **Duration**, Microsoft® Schedule+ interprets it as lasting exactly 24 hours. That is, Schedule+ treats **Duration** values of 0, 1440, 2880, 4320, and so on as if they were 1440.

Changes you make to properties on a RecurrencePattern object take effect when you call the underlying appointment's Send or Update method.

EndTime Property (RecurrencePattern Object) Group

The **EndTime** property returns or sets the ending date/time for each recurrence of the appointment. Read/write.

Syntax

objRecurPatt.**EndTime**

Data Type

Variant (**vbDate** format)

Remarks

The **EndTime** property contains the time at which the appointment is to terminate every time it recurs. **EndTime** is always valid on a newly created RecurrencePattern object and defaults to the **EndTime** property of the AppointmentItem object that created this recurrence pattern.

The **EndTime** property ignores seconds and truncates the time component to the minute.

EndTime uses both the date and the time component of the **vbDate** format. When you read **EndTime**, you get the date/time of the end of the original appointment.

If the time component of **EndTime** is earlier than StartTime, each occurrence is treated as ending on the day after it starts.

The recurrence pattern's **StartTime** and **EndTime** properties are always held internally in the organizer's current time zone. By contrast, the AppointmentItem object's **StartTime** and **EndTime** properties are always held internally in UTC (Coordinated Universal Time, also known as GMT). However, all these properties are converted to the local messaging user's current time zone whenever they are displayed or read programmatically.

Setting the **EndTime** property causes CDO to force certain other recurrence pattern properties into conformance. Duration is recalculated from **StartTime** and **EndTime**.

Changes you make to properties on a RecurrencePattern object take effect when you call the underlying appointment's Send or Update method.

Instance Property (RecurrencePattern Object) Group

The **Instance** property returns or sets the instance of the day within the month on which the appointment recurs. Read/write.

Syntax

objRecurPatt.Instance

Data Type

Long

Remarks

The **Instance** property is used when the AppointmentItem is to recur only once during each recurrence unit, such as the second Wednesday of every month or the first Tuesday of every January. The DayOfWeekMask property must specify exactly one day of the week, and **Instance** selects the occurrence of that day within the month on which recurrence is enabled. The last occurrence of the day within the month can be represented by the value 5.

Instance is only valid if the value of the RecurrenceType property is **CdoRecurTypeMonthlyNth** or **CdoRecurTypeYearlyNth**. When **Instance** is valid on a newly created RecurrencePattern object, it defaults to the current instance of the current day of the week. For example, if the RecurrencePattern object is created on Wednesday 9 December and **RecurrenceType** is set to **CdoRecurTypeYearlyNth**, then **DayOfWeekMask** defaults to 8 (**CdoWednesday**), **Instance** defaults to 2 (the second Wednesday of the month), and MonthOfYear defaults to 12 (December).

Changes you make to properties on a RecurrencePattern object take effect when you call the underlying appointment's Send or Update method.

Example

This code fragment uses the **Instance** and Interval properties to specify that an appointment is to recur on the third Sunday of every other month:

```
Dim objAppointment As AppointmentItem
Dim objRecurrence As RecurrencePattern
' ... assume AppointmentItem object is valid ...
Set objAppointment = objAppointments.Add
With objAppointment
    .Subject = "Using Instance and Interval with MonthlyNth"
    .Text = "Creating an appointment and making it recur on the " _
        & "3rd Sunday of every other month for one year."
    .Location = "My office"
Set objRecurrence = .GetRecurrencePattern
With objRecurrence
    .RecurrenceType = CdoRecurTypeMonthlyNth
    .PatternStartDate = Now
    .PatternEndDate = DateAdd("m", 12, .PatternStartDate)
    .StartTime = .PatternStartDate ' takes only the time component
    .EndTime = DateAdd("h", 1, .StartTime) ' 1-hour appointment
    .DayOfWeekMask = 1 ' CdoSunday
    .Instance = 3 ' third Sunday of month
    .Interval = 2 ' every other month
End With
objAppointment.Update ' must do this for settings to take effect
```

End With

Interval Property (RecurrencePattern Object) Group

The **Interval** property returns or sets the number of recurrence units between recurrences of the appointment. Read/write.

Syntax

objRecurPatt.Interval

Data Type

Long

Remarks

The **Interval** property is used when the AppointmentItem is to recur less often than every recurrence unit, such as once every three days, once every two weeks, or once every six months. **Interval** contains a value representing the frequency of recurrence in terms of recurrence units.

Interval is always valid on a newly created RecurrencePattern object and defaults to 1, which is its minimum value. Its maximum value depends on the setting of the RecurrenceType property as follows:

| RecurrenceType setting | Maximum Interval value |
|------------------------|------------------------|
| CdoRecurTypeDaily | 999 |
| CdoRecurTypeMonthly | 99 |
| CdoRecurTypeMonthlyNth | |
| CdoRecurTypeYearly | 1 |
| CdoRecurTypeYearlyNth | |
| CdoRecurTypeWeekly | 99 |

Setting the **Interval** property causes CDO to force certain other recurrence pattern properties into conformance. **PatternEndDate** is recalculated from **PatternStartDate**, **Occurrences**, and **Interval**. If the resulting **PatternEndDate** is January 1, 4000 or later, **NoEndDate** is automatically reset to **True**, **Occurrences** is reset to 1,490,000, **PatternEndDate** is reset to the month and day of **PatternStartDate** in the year 4001, and the recurrence pattern is considered to extend infinitely far into the future.

Changes you make to properties on a RecurrencePattern object take effect when you call the underlying appointment's **Send** or **Update** method.

MonthOfYear Property (RecurrencePattern Object)

Group

The **MonthOfYear** property returns or sets the month of the year in which the appointment recurs. Read/write.

Syntax

objRecurPatt.MonthOfYear

Data Type

Long

Remarks

The **MonthOfYear** property contains the calendar number of the month in which the AppointmentItem is to recur, for example the value 2 to indicate February.

MonthOfYear is only valid if the value of the RecurrenceType property is **CdoRecurTypeYearly** or **CdoRecurTypeYearlyNth**. When **MonthOfYear** is valid on a newly created RecurrencePattern object, it defaults to the current month.

Note If the AppointmentItem object has been made into a meeting, the DayOfMonth, DayOfWeekMask, and **MonthOfYear** properties are all held internally in the meeting organizer's current time zone. If these properties are displayed or read by a messaging user in a different time zone, they are not converted. The user or application accessing these properties may need to be aware that they represent the organizer's time zone.

Changes you make to properties on a RecurrencePattern object take effect when you call the underlying appointment's Send or Update method.

NoEndDate Property (RecurrencePattern Object)

Group

The **NoEndDate** property indicates whether the recurrence pattern has an ending date. Read/write.

Syntax

objRecurPatt.NoEndDate

Data Type

Boolean

Remarks

The **NoEndDate** property contains **True** if the AppointmentItem object is to recur indefinitely and **False** if the recurrence has a terminal date. **NoEndDate** is always valid on a newly created RecurrencePattern object and defaults to **True**.

Setting the **NoEndDate** property causes CDO to force certain other recurrence pattern properties into conformance. If **NoEndDate** is set to **True**, **Occurrences** is reset to 1,490,000 and **PatternEndDate** is reset to the month and day of **PatternStartDate** in the year 4001. If **NoEndDate** is set to **False**, **Occurrences** is reset to 10 and **PatternEndDate** is recalculated from **PatternStartDate** and **Occurrences**. However, no changes are made if **NoEndDate** was already **False**.

CDO imposes a limit of December 31, 3999 on **PatternEndDate** and 1,489,999 on **Occurrences**. If either of these properties exceeds its limit, **NoEndDate** is automatically reset to **True**, **Occurrences** is reset to 1,490,000, **PatternEndDate** is reset to the month and day of **PatternStartDate** in the year 4001, and the recurrence pattern is considered to extend infinitely far into the future.

Changes you make to properties on a RecurrencePattern object take effect when you call the underlying appointment's Send or Update method.

Occurrences Property (RecurrencePattern Object)

Group

The **Occurrences** property returns or sets the number of occurrences of the recurrence pattern. Read/write.

Syntax

objRecur Patt.Occurrences

Data Type

Long

Remarks

The **Occurrences** property is used when the appointment is to recur a specific number of times, such as the next ten Thursdays. **Occurrences** has a minimum value of 1 and represents the total number of occurrences of the AppointmentItem object fitting the recurrence pattern. This qualification is necessary because the days of the original appointment, the PatternStartDate, and the PatternEndDate are not required to be included in the pattern specified by the DayOfMonth, DayOfWeekMask, Instance, Interval, and MonthOfYear properties. The original appointment is counted in **Occurrences** only if it matches the pattern.

Occurrences is always valid on a newly created RecurrencePattern object and defaults to its maximum value of 1,490,000. If the NoEndDate property is subsequently set to **False**, **Occurrences** defaults to 10.

Setting the **Occurrences** property causes CDO to force certain other recurrence pattern properties into conformance. PatternEndDate is recalculated from PatternStartDate and **Occurrences**. **NoEndDate** is reset to **False** if **Occurrences** is less than 1,490,000, or to **True** if **Occurrences** is 1,490,000 or greater or the recalculated PatternEndDate is January 1, 4000 or later.

CDO imposes a limit of December 31, 3999 on PatternEndDate and 1,489,999 on **Occurrences**. If either of these properties exceeds its limit, **NoEndDate** is automatically reset to **True**, **Occurrences** is reset to 1,490,000, PatternEndDate is reset to the month and day of PatternStartDate in the year 4001, and the recurrence pattern is considered to extend infinitely far into the future.

Changes you make to properties on a RecurrencePattern object take effect when you call the underlying appointment's Send or Update method.

PatternEndDate Property (RecurrencePattern Object)

Group

The **PatternEndDate** property returns or sets the day on or before which the appointment last recurs. Read/write.

Syntax

objRecurPatt.PatternEndDate

Data Type

Variant (**vbDate** format)

Remarks

The **PatternEndDate** property contains the latest possible date of the last occurrence of the appointment. This qualification is necessary because **PatternEndDate** is not required to be included in the pattern specified by the **DayOfMonth**, **DayOfWeekMask**, **Instance**, **Interval**, and **MonthOfYear** properties. A recurrence is generated on **PatternEndDate** only if it matches the pattern.

PatternEndDate is always valid on a newly created RecurrencePattern object and defaults to the month and day of the **PatternStartDate** property in the year 4001.

The time component of the **vbDate** format is ignored when you set the **PatternEndDate** property. When you read **PatternEndDate**, the time component of the **StartTime** property is used to return a full **vbDate** value representing the starting time on the ending date.

The date component of **PatternEndDate** must not be earlier than **PatternStartDate**. These two properties determine the overall time period during which the **AppointmentItem** object is scheduled for recurrence.

Setting the **PatternEndDate** property causes CDO to force certain other recurrence pattern properties into conformance. **Occurrences** is recalculated from **PatternStartDate** and **PatternEndDate**, and **NoEndDate** is reset to **False**. However, if **PatternEndDate** is January 1, 4000 or later, **NoEndDate** is reset to **True**, **Occurrences** is reset to 1,490,000, and **PatternEndDate** is reset to the month and day of **PatternStartDate** in the year 4001.

You cannot change **PatternEndDate** to a date earlier than the current **PatternStartDate**.

If you change the **PatternEndDate** after one or more individual recurrences have been instantiated, Microsoft® Schedule+ automatically deletes any recurrences later than the new **PatternEndDate**, and Microsoft® Outlook™ deletes all instantiated recurrences. CDO determines what your active calendar store is and deletes recurrences in the appropriate manner.

CDO imposes a limit of December 31, 3999 on **PatternEndDate** and 1,489,999 on **Occurrences**. If either of these properties exceeds its limit, **NoEndDate** is automatically reset to **True**, **Occurrences** is reset to 1,490,000, **PatternEndDate** is reset to the month and day of **PatternStartDate** in the year 4001, and the recurrence pattern is considered to extend infinitely far into the future.

Changes you make to properties on a RecurrencePattern object take effect when you call the underlying appointment's **Send** or **Update** method.

PatternStartDate Property (RecurrencePattern Object)

Group

The **PatternStartDate** property returns or sets the day on or after which the appointment first recurs. Read/write.

Syntax

objRecur Patt.PatternStartDate

Data Type

Variant (**vbDate** format)

Remarks

The **PatternStartDate** property contains the earliest possible date of the first occurrence of the appointment. This qualification is necessary because **PatternStartDate** is not required to be included in the pattern specified by the **DayOfMonth**, **DayOfWeekMask**, **Instance**, **Interval**, and **MonthOfYear** properties. A recurrence is generated on **PatternStartDate** only if it matches the pattern.

PatternStartDate is always valid on a newly created RecurrencePattern object and defaults to the date component of the **StartTime** property of the **AppointmentItem** object that created this recurrence pattern.

The time component of the **vbDate** format is ignored when you set the **PatternStartDate** property. When you read **PatternStartDate**, the time component of the **StartTime** property is used to return a full **vbDate** value representing the starting time on the starting date.

The date component of **PatternStartDate** must not be later than **PatternEndDate**. These two properties determine the overall time period during which the **AppointmentItem** object is scheduled for recurrence.

Setting the **PatternStartDate** property causes CDO to force certain other recurrence pattern properties into conformance. **PatternEndDate** is recalculated from **PatternStartDate** and **Occurrences**. However, if the new **PatternEndDate** is January 1, 4000 or later, **NoEndDate** is reset to **True**, **Occurrences** is reset to 1,490,000, and **PatternEndDate** is reset to the month and day of **PatternStartDate** in the year 4001.

If you change **PatternStartDate** to a date later than the current **PatternEndDate**, **PatternEndDate** is recalculated using the current value of **Occurrences**. Since **PatternEndDate** is always forced to be no earlier than **PatternStartDate**, **Occurrences** always has a minimum value of 1.

If you change the **PatternStartDate** after one or more individual recurrences have been instantiated, Microsoft® Schedule+ automatically deletes any recurrences earlier than the new **PatternStartDate**, and Microsoft® Outlook™ deletes all instantiated recurrences. CDO determines what your active calendar store is and deletes recurrences in the appropriate manner.

CDO imposes a limit of December 31, 3999 on **PatternStartDate**. If you attempt to exceed this limit, an error is returned.

Changes you make to properties on a RecurrencePattern object take effect when you call the underlying appointment's **Send** or **Update** method.

RecurrenceType Property (RecurrencePattern Object)

Group

The **RecurrenceType** property returns or sets the recurrence unit and the frequency with which the appointment recurs. Read/write.

Syntax

objRecurPatt.RecurrenceType

Data Type

Long

Remarks

The **RecurrenceType** property determines the periodicity with which the [AppointmentItem](#) is to recur. It can have exactly one of the following values:

| RecurrenceType setting | Decimal value | Other properties used in conjunction with this setting |
|------------------------|---------------|--|
| CdoRecurTypeDaily | 0 | Interval |
| CdoRecurTypeWeekly | 1 | DayOfWeekMask
Interval |
| CdoRecurTypeMonthly | 2 | DayOfMonth
Interval |
| CdoRecurTypeMonthlyNth | 3 | DayOfWeekMask
Instance
Interval |
| CdoRecurTypeYearly | 5 | DayOfMonth
Interval
MonthOfYear |
| CdoRecurTypeYearlyNth | 6 | DayOfWeekMask
Instance
Interval
MonthOfYear |

RecurrenceType is always valid on a newly created RecurrencePattern object and defaults to **CdoRecurTypeWeekly**.

If you instantiate one or more individual recurrences and subsequently delete them all, Microsoft® Outlook™ deletes the RecurrencePattern object. If CDO determines that your active calendar store is Outlook, it also deletes the RecurrencePattern object in this case.

Changes you make to properties on a RecurrencePattern object take effect when you call the underlying appointment's **Send** or **Update** method.

StartTime Property (RecurrencePattern Object) Group

The **StartTime** property returns or sets the starting date/time for each recurrence of the appointment. Read/write.

Syntax

objRecurPatt.**StartTime**

Data Type

Variant (**vbDate** format)

Remarks

The **StartTime** property contains the time at which the appointment is to begin every time it recurs. **StartTime** is always valid on a newly created RecurrencePattern object and defaults to the time component of the **StartTime** property of the AppointmentItem object that created this recurrence pattern.

The **StartTime** property ignores seconds and truncates the time component to the minute.

StartTime uses both the date and the time component of the **vbDate** format. When you set **StartTime**, the date component must match the date component of the **PatternStartDate** property, or **CdoE_INVALID_PARAMETER** is returned. When you read **StartTime**, you get the date/time of the start of the original appointment.

If the time component of **EndTime** is earlier than **StartTime**, each occurrence is treated as ending on the day after it starts.

The recurrence pattern's **StartTime** and **EndTime** properties are always held internally in the organizer's current time zone. By contrast, the AppointmentItem object's **StartTime** and **EndTime** properties are always held internally in UTC (Coordinated Universal Time, also known as GMT). However, all these properties are converted to the local messaging user's current time zone whenever they are displayed or read programmatically.

Setting the **StartTime** property causes CDO to force certain other recurrence pattern properties into conformance. **Duration** is recalculated from **StartTime** and **EndTime**.

Changes you make to properties on a RecurrencePattern object take effect when you call the underlying appointment's **Send** or **Update** method.

Session Object

The Session object contains session-wide settings and options. It also contains properties that return top-level objects, such as **CurrentUser**.

At a Glance

| | |
|----------------------------|---|
| Specified in type library: | CDO.DLL |
| First available in: | CDO Library version 1.0.a |
| Parent objects: | (none) |
| Child objects: | <u>AddressLists</u> collection
<u>Folder</u> (Inbox or Outbox)
<u>InfoStores</u> collection |
| Default property: | <u>Name</u> |

Properties

| Name | Available since version | Type | Access |
|------------------------|-------------------------|--|---|
| <u>AddressLists</u> | 1.1 | AddressList object or AddressLists collection object | Read-only |
| <u>Application</u> | 1.0.a | String | Read-only |
| <u>Class</u> | 1.0.a | Long | Read-only |
| <u>CurrentUser</u> | 1.0.a | AddressEntry object | Read-only |
| <u>Inbox</u> | 1.0.a | Folder object | Read-only |
| <u>InfoStores</u> | 1.0.a | InfoStore object or InfoStores collection object | Read-only |
| <u>MAPIOBJECT</u> | 1.0.a | IUnknown object | Read/write (Note: Not available to Visual Basic applications) |
| <u>Name</u> | 1.0.a | String | Read-only |
| <u>OperatingSystem</u> | 1.0.a | String | Read-only |
| <u>Outbox</u> | 1.0.a | Folder object | Read-only |
| <u>OutOfOffice</u> | 1.1 | Boolean | Read/write |
| <u>OutOfOfficeText</u> | 1.1 | String | Read/write |
| <u>Parent</u> | 1.0.a | Object; set to Nothing | Read-only |
| <u>Session</u> | 1.0.a | Session object (itself) | Read-only |
| <u>Version</u> | 1.0.a | String | Read-only |

Methods

Available

| Name | since version | Parameters |
|---------------------------------------|----------------------|---|
| <u>AddressBook</u> | 1.0.a | (optional) <i>recipients</i> as Object ,
(optional) <i>title</i> as String ,
(optional) <i>oneAddress</i> as Boolean ,
(optional) <i>forceResolution</i> as Boolean ,
(optional) <i>recipLists</i> as Long ,
(optional) <i>toLabel</i> as String ,
(optional) <i>ccLabel</i> as String ,
(optional) <i>bccLabel</i> as String ,
(optional) <i>parentWindow</i> as Long |
| <u>CompareIDs</u> | 1.1 | <i>ID1</i> as String ,
<i>ID2</i> as String |
| <u>CreateConversationIndex</u> | 1.1 | (optional) <i>ParentIndex</i> as String |
| <u>DeliverNow</u> | 1.1 | (none) |
| <u>GetAddressEntry</u> | 1.0.a | <i>entryID</i> as String |
| <u>GetAddressList</u> | 1.2 | <i>ObjectType</i> as Long |
| <u>GetArticle</u> | 1.2 | <i>ArticleID</i> as Long ,
<i>FolderID</i> as String ,
(optional) <i>StoreID</i> as String |
| <u>GetDefaultFolder</u> | 1.2 | <i>ObjectType</i> as Long |
| <u>GetFolder</u> | 1.0.a | <i>folderID</i> as String ,
(optional) <i>storeID</i> as String |
| <u>GetInfoStore</u> | 1.0.a | (optional) <i>storeID</i> as String |
| <u>GetMessage</u> | 1.0.a | <i>messageID</i> as String ,
(optional) <i>storeID</i> as String |
| <u>GetOption</u> | 1.2 | <i>OptType</i> as String |
| <u>Logoff</u> | 1.0.a | (none) |
| <u>Logon</u> | 1.0.a | (optional) <i>profileName</i> as String ,
(optional) <i>profilePassword</i> as String ,
(optional) <i>showDialog</i> as Boolean ,
(optional) <i>newSession</i> as Boolean ,
(optional) <i>parentWindow</i> as Long ,
(optional) <i>NoMail</i> as Boolean ,
(optional) <i>ProfileInfo</i> as String |
| <u>SetLocaleIDs</u> | 1.1 | <i>LocaleID</i> as Long ,
<i>CodePageID</i> as Long |
| <u>SetOption</u> | 1.2 | <i>OptType</i> as String ,
<i>OptValue</i> as Variant |

Remarks

A Session object is considered a top-level object, meaning it can be created directly from a Microsoft® Visual Basic® program. In the CDO for Exchange Library it has a ProgID of MAPI.Session. This code

fragment creates a Session object through early binding:

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

This code fragment creates a Session object through late binding:

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

Generally, early binding is preferable, because it enforces type checking and generates more efficient code. Note that you specify the full ProgID "MAPI.Session" instead of just "Session" in order to distinguish a MAPI session from other types of sessions available to a Visual Basic program through other object libraries.

In both cases, after you create a new Session object, you use the **Logon** method to initiate a session with MAPI. No other activities with CDO are permitted prior to a successful logon. The only exception to this rule is the Session object's **SetLocaleIDs** method.

AddressBook Method (Session Object)

Group

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*])

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 initial values for the recipient list boxes in the address book dialog box. During the dialog, the user can select recipients from this collection and add other recipients.

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 at a time. 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 setting

Action

| | |
|----|--|
| -1 | Displays three list boxes with default captions and without resolution, that is, acts as a shortcut for <i>forceResolution=False</i> , <i>recipLists=3</i> with no setting of <i>toLabel</i> , <i>ccLabel</i> , or <i>bccLabel</i> . |
| 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 CdoTo recipients (default). |
| 2 | Displays two list boxes for CdoTo and CdoCc recipients. |
| 3 | Displays three list boxes for CdoTo , CdoCc , and CdoBcc recipients. |

toLabel

Optional. String. The caption for the button associated with the first recipient 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 recipient 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 recipient 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 box should be application-modal.

Remarks

The **AddressBook** method returns **CdoE_USER_CANCEL** if the user cancels the dialog box.

The *recipients* parameter provides initial values for the recipient list boxes. These values expedite the user's recipient selection process. A common use of this parameter is to set it to the Recipients collection of a message to which you are generating a reply.

When you use **AddressBook** to let the user select recipients for a new message, you use either two or three different Recipients collections, depending on whether you furnish the *recipients* parameter. Use the following procedure:

1. Optionally, prepare an initial Recipients collection to be submitted in the *recipients* parameter to the **AddressBook** method.
2. Call **AddressBook**, which returns the user-selected Recipients collection.
3. Call the **Add** method on a Messages collection to create a new message.
4. 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 *oneAddress* parameter indicates whether only one address entry at a time can be selected before being added to the recipients list. If *oneAddress* is set to **False**, the user can select multiple recipients by using the CTRL or SHIFT key during the selection. If *oneAddress* is set to **True**, multiple selection is disabled.

To provide an access key for the recipient 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.

The address book dialog box is always modal, meaning the parent window is disabled while the dialog box 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 box is active. If the *parentWindow* parameter is supplied but is not valid, the call returns **CdoE_INVALID_PARAMETER**.

The following methods can invoke dialog boxes:

- **Details** method (AddressEntry object)
- **Options** and **Send** methods (Message object)
- **Resolve** method (Recipient object)
- **Resolve** method (Recipients collection)
- **AddressBook** and **Logon** methods (Session object)

However, if your application is running as a Microsoft® Windows NT® service, for example from Active Server Pages (ASP) script on a Microsoft® Internet Information Server (IIS), no user interface is allowed.

For more information on Windows NT services, see the Win32® Web page *Using MAPI from a Windows NT Service* at <http://www.microsoft.com/win32dev/mapi/mapiserv.htm>. For more information

on running as a service, see "Windows NT Service Client Applications" in the *MAPI Programmer's Reference*.

Example

This 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 log on"
    Exit Function
End If
Set objRecipColl = objSession.AddressBook( _
    Title:="Select Attendees", _
    forceResolution:=True, _
    recipLists:=3, _
    toLabel:="&Very Important People", _ ' on button
    ccLabel:="&Fairly Important People", _
    bccLabel:="&Secret Important People")
' "recipients:=" parameter not used in preceding call
MsgBox "Name of first recipient = " & objRecipColl.Item(1).Name
' could be objRecipColl(1) since Item and Name are default properties
Exit Function
```

AddressLists Property (Session Object)

Group

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*)

Set *objOneAddrList* = *objSession*.**AddressLists**(*name*)

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.

name

String. The value of the **Name** property of the AddressList object to be selected.

Data Type

Object (AddressList or AddressLists collection)

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.

Although the **AddressLists** property itself is read-only, the collection it returns can be accessed in the normal manner, and the properties on its member [AddressList](#) objects retain their respective read/write or read-only accessibility.

CompareIDs Method (Session Object)

Group

The **CompareIDs** method determines whether two CDO Library objects are the same object.

Syntax

objSession.**CompareIDs**(*ID1*, *ID2*)

objSession

Required. The Session object.

ID1

Required. String. The unique identifier of the first object to be compared.

ID2

Required. String. The unique identifier of the second object to be compared.

Remarks

The **CompareIDs** method compares the identifiers of two arbitrary CDO 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 in the underlying messaging system. 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**.

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. However, MAPI does not require identifier values to be binary comparable. Accordingly, two identifier values can be different, yet refer to the same object. For more information on entry identifiers, see the *MAPI Programmer's Reference*.

The **CompareIDs** method ultimately calls one of the MAPI **CompareEntryIDs** methods to determine if two objects are the same. Several CDO Library objects also provide the **IsSameAs** method for a comparison of two objects of that particular type.

CreateConversationIndex Method (Session Object) Group

The **CreateConversationIndex** method creates or updates an index for a conversation thread.

Syntax

```
newIndex = objSession.CreateConversationIndex( [ParentIndex] )
```

newIndex

String. The conversation index to be assigned to the first or next message in the thread.

objSession

Required. The Session object.

ParentIndex

Optional. String. The conversation index of a received message for which a reply is being generated.

Remarks

The **CreateConversationIndex** method takes the current value of a conversation index and returns a new value suitable for an outgoing message. If the message to be sent represents the beginning of a new thread, there is no current index, and the *ParentIndex* parameter is not passed in. If the outgoing message is a reply to a received message, the ConversationIndex property of the received message is passed in as the *ParentIndex* parameter.

The **CreateConversationIndex** method ultimately calls the MAPI **ScCreateConversationIndex** function.

Example

This code fragment responds to the first message in the Inbox:

```
Dim objInMsgColl As Messages ' messages in Inbox
Dim objRecMsg As Message    ' received message
Dim objNewMsg As Message    ' outgoing reply message
Dim objRcpColl As Recipients ' recipients for outgoing message
Dim strSenderId As String   ' unique ID of original sender
Dim strCnvIdx As String     ' conversation index
```

```
If objSession Is Nothing Then
    MsgBox "Must first create MAPI session and log on"
    Exit Function
End If
Set objInMsgColl = objSession.Inbox.Messages
' ... error handling ...
If objInMsgColl Is Nothing Then
    MsgBox "Could not successfully access Inbox messages"
    Exit Function
End If
Set objRecMsg = objInMsgColl.GetFirst()
If objRecMsg Is Nothing Then
    MsgBox "No messages in Inbox"
    Exit Function
End If
' make new conversation index from old
strCnvIdx = objSession.CreateConversationIndex _
            (objRecMsg.ConversationIndex)
```



```
strSenderID = objRecMsg.Sender.ID ' save sender's unique ID
Set objNewMsg = objSession.Outbox.Messages.Add ' generate reply
' ... error handling ...
Set objRcpColl = objNewMsg.Recipients
' ... error handling ...
objRcpColl.Add entryID:=strSenderID ' add sender as recipient
' ... error handling ...
With objNewMsg
    .ConversationIndex = strCnvIndx
    .ConversationTopic = objRecMsg.ConversationTopic
    .Subject = "RE: " & objRecMsg.Subject
    .Text = "Please consider this a reply to your message."
    .Update ' save everything in the MAPI system
    .Send showDialog:=False
End With
```

CurrentUser Property (Session Object)

Group

The **CurrentUser** property returns the active user as an [AddressEntry](#) object. Read-only.

Syntax

Set *objAddrEntry* = *objSession*.**CurrentUser**

objAddrEntry

Object. The returned AddressEntry object that represents the messaging user logged on to the session.

objSession

Object. The Session object.

Data Type

Object (AddressEntry)

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)

Group

The **DeliverNow** method requests immediate delivery of all undelivered messages submitted in the current session.

Syntax

objSession.**DeliverNow**()

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) Group

The **GetAddressEntry** method returns an [AddressEntry](#) object.

Syntax

Set *objAddressEntry* = *objSession*.**GetAddressEntry**(*entryID*)

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
```

GetAddressList Method (Session Object)

Group

The **GetAddressList** method returns an [AddressList](#) object from a directory service.

Syntax

Set *objAddressList* = *objSession*.**GetAddressList**(*ObjectType*)

objAddressList

On successful return, represents the default AddressList object specified by *ObjectType*.

objSession

Required. The Session object.

ObjectType

Required. Long. Specifies the address list to be retrieved.

Remarks

The **GetAddressList** method returns the default address list of the specified type for the default directory service of the current session.

The *ObjectType* parameter can have exactly one of the following values:

| <i>ObjectType</i> setting | Value | Default Address list retrieved |
|----------------------------------|--------------|---------------------------------------|
| CdoAddressListGAL | 0 | Global address list |
| CdoAddressListPAB | 1 | Personal address book |

GetArticle Method (Session Object)

Group

The **GetArticle** method returns a [Message](#) object corresponding to an article number.

Syntax

Set *objMsg* = *objSession*.**GetArticle**(*ArticleID*, *FolderID* [, *StoreID*])

objMsg

On successful return, represents the Message object.

objSession

Required. The Session object.

ArticleID

Required. Long. Specifies the article number of the message to be retrieved.

FolderID

Required. String. Specifies the unique identifier of the folder from which to obtain the message.

StoreID

Optional. String. Specifies the unique identifier of the message store containing the folder that contains the message. *StoreID* defaults to the current session's default message store.

Remarks

The **GetArticle** method retrieves any [Message](#) object for which you can supply an article number. You do not have to know the message's **ID** property.

An article number is a long integer associated with an item in a message store. The store computes and sets the article number on each item as it is added to a folder. Every item in every folder is assigned an article number, regardless of its **Type** property, that is, its message class.

Not all message stores support article numbers. If you call **GetArticle** specifying a store that does not support them, **CdoE_NOT_FOUND** is returned.

The advantage of an article number is that it is only 32 bits long, as opposed to the item's unique identifier in its **ID** property, which is typically on the order of 70 bytes. If you are constructing a URL to reference the item, you can make the URL appreciably shorter by using the item's article number instead of its unique identifier.

Unlike a unique identifier, an article number is only unique within a single folder. Therefore, unlike the **GetMessage** method, **GetArticle** requires that you specify the folder with the *FolderID* parameter. The item to be retrieved must also be in the normal **Messages** collection of the folder. If you specify the article number of an item in the **HiddenMessages** collection, **GetArticle** returns **CdoE_NOT_FOUND**.

An article number should not be confused with the **Index** property available on members of several CDO collections. An article number is assigned once and changes only when the item is modified with the **Update** method, while the **Index** property on a member of a collection can change when other members are added or deleted, or when the collection is sorted.

The **GetArticle** method can be used to decode a URL that refers to a message by article number. Such a URL is typically of the form

```
http://server/virtroot/forms/openitem.asp?folder=<foldername>&art=<articlenum>
```

The unique identifier of a [Folder](#) object can be obtained from its **ID** property. A message store's unique identifier is returned by the [InfoStore](#) object's **ID** property.

Performance is likely to be improved if you supply the *StoreID* parameter.

The article number of an item in a folder corresponds to the MAPI property PR_INTERNET_ARTICLE_NUMBER on that item. When you are constructing a URL, you can use the Message object's **Fields** property to obtain its article number:

```
Dim article As Long ' article number of message
article = objMessage.Fields(CdoPR_INTERNET_ARTICLE_NUMBER)
```

You can use this approach to see if the message store you are dealing with supports article numbers, because if it does, it assigns them to every item in every folder.

GetDefaultFolder Method (Session Object) Group

The **GetDefaultFolder** method returns a Folder object from a message store.

Syntax

Set *objFolder* = *objSession*.**GetDefaultFolder**(*ObjectType*)

objFolder

On successful return, represents the default Folder object specified by *ObjectType*.

objSession

Required. The Session object.

ObjectType

Required. Long. Specifies the default folder to be retrieved.

Remarks

The **GetDefaultFolder** method returns the default folder of the specified type for the default message store of the current session.

The *ObjectType* parameter can have exactly one of the following values:

| ObjectType setting | Value | Default folder retrieved |
|-------------------------------------|--------------|---------------------------------|
| CdoDefaultFolderCalendar | 0 | Calendar |
| CdoDefaultFolderContacts | 5 | Contacts |
| CdoDefaultFolderDeletedItems | 4 | Deleted Items |
| CdoDefaultFolderInbox | 1 | Inbox |
| CdoDefaultFolderJournal | 6 | Journal |
| CdoDefaultFolderNotes | 7 | Notes |
| CdoDefaultFolderOutbox | 2 | Outbox |
| CdoDefaultFolderSentItems | 3 | Sent Items |
| CdoDefaultFolderTasks | 8 | Tasks |

The Contacts, Journal, Notes, and Tasks folders are specific to Microsoft® Outlook™. If your application is running in a purely Microsoft® Schedule+ environment, an attempt to access any of these four folders returns **CdoE_NOT_FOUND**.

Not all message store providers support all folder types. A personal message store (PST), for example, typically does not support a calendar folder. If your profile specifies only a PST and you call **GetDefaultFolder** specifying **CdoDefaultFolderCalendar**, you may get a return of **CdoE_NO_SUPPORT**.

GetFolder Method (Session Object)

Group

The **GetFolder** method returns a Folder object from a MAPI message store.

Syntax

Set *objFolder* = *objSession*.**GetFolder**(*folderID* [, *storeID*])

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 **CdoE_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.

If your application is running as a Microsoft® Windows NT® service, you cannot access the Microsoft Exchange Public Folders through the normal hierarchy because of a notification conflict. You must use the InfoStore object's **Fields** property to obtain the Microsoft Exchange property PR_IPM_PUBLIC_FOLDERS_ENTRYID, property tag &H66310102. This represents the top-level public folder and allows you to access all other public folders through its **Folders** property. For more information on Windows NT services, see the Win32® Web page *Using MAPI from a Windows NT Service* at <http://www.microsoft.com/win32dev/mapi/mapiserv.htm>.

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)

Group

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*)

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

Optional. String. Specifies the unique identifier of the InfoStore object to retrieve. If *storeID* is an empty string or is not supplied, the default InfoStore for the session is returned.

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 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 InfoStore
    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)

Group

The **GetMessage** method returns an [AppointmentItem](#), [MeetingItem](#), or [Message](#) object from a MAPI message store.

Syntax

Set *objMessage* = *objSession*.**GetMessage**(*messageID* [, *storeID*])

objMessage

On successful return, contains the [AppointmentItem](#), [MeetingItem](#), or [Message](#) object with the specified identifier. If the specified *messageID* does not exist, **GetMessage** returns **CdoE_NOT_FOUND**.

objSession

Required. The Session object.

messageID

Required. String. Specifies the unique identifier of the appointment, meeting, or 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 [AppointmentItem](#), [MeetingItem](#), or [Message](#) 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
```

GetOption Method (Session Object)

Group

The **GetOption** method returns a calendar rendering option for the session.

Syntax

objSession.**GetOption**(*OptType*)

objSession

Required. The Session object.

OptType

Required. String. Selects the option by name.

Remarks

The calendar rendering options are set with the **SetOption** method. A CDO Rendering application can use their values to set the corresponding properties of a ContainerRenderer object when the rendering application logs on to the session. The container renderer properties are used by the **RenderAppointments**, **RenderDateNavigator**, and **RenderEvents** methods of a CalendarView object to render calendar data. The CDO Rendering ObjectRenderer object can also use the value of the "TimeZone" option.

Note that the rendering objects do not automatically assimilate the values of the Session object's calendar rendering options. You must transfer these values yourself between the Session object and the appropriate properties of the rendering object you are using.

GetOption and **SetOption** return **CdoE_CALL_FAILED** if you call them while you are not logged on to the session. Following a successful logon, **GetOption** returns the option settings that were stored in the messaging user's Inbox with **SetOption**. If no options have ever been stored for the messaging user, the default settings are returned.

The "CalendarStore" option can be "Outlook" or "SchedulePlus", depending on the store underlying the messaging user's calendar folder. This option is useful if you have both types of calendar store and need to determine which one is active. It defaults to "Outlook" if you have neither type or both types. Calendar store information is irrelevant to the container renderer, so "CalendarStore" has no corresponding rendering object property.

The calendar rendering options that can be selected are as follows:

| <i>OptType</i> string value | Data type | Corresponding <u>ContainerRenderer</u> property | Default value |
|------------------------------------|---------------------------------|--|-------------------------------------|
| "BusinessDayEndTime" | Variant (vbDate format) | <u>BusinessDayEndTime</u> | 5:00 PM (17:00) |
| "BusinessDayStartTime" | Variant (vbDate format) | <u>BusinessDayStartTime</u> | 9:00 AM (09:00) |
| "CalendarStore" | String | (none) | Current calendar store or "Outlook" |
| "FirstDayOfWeek" | Long | <u>FirstDayOfWeek</u> | 7 (Sunday) |
| "Is24HourClock" | Boolean | <u>Is24HourClock</u> | False |

| | | | |
|---------------|------|---------------------|---|
| "TimeZone" | Long | <u>TimeZone</u> | Current zone on Web server |
| "WorkingDays" | Long | <u>BusinessDays</u> | 62
(CdoMonday ... CdoFriday) |

The session's options and the container renderer's properties are not automatically kept in synchronization. Changing an option does not cause the corresponding property to be changed, nor vice versa. Your application is responsible for setting all options and properties as appropriate.

Note that the "FirstDayOfWeek" option and the FirstDayOfWeek property are compatible with each other in using an enumeration starting with 1 for Monday and ending with 7 for Sunday, but they are not compatible with the DayOfWeekMask property of the RecurrencePattern object, which uses the mask constants **CdoSunday** through **CdoSaturday**. The "WorkingDays" option, however, is compatible with the mask constants.

Inbox Property (Session Object)

Group

The **Inbox** property returns a Folder object representing the current user's Inbox folder. Read-only.

Syntax

objSession.Inbox

Data Type

Object (Folder)

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, CDO 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:

```
' 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)

Group

The **InfoStores** property returns a single InfoStore object or an InfoStores collection available to this session. Read-only.

Syntax

Set *objInfoStoresColl* = *objSession*.**InfoStores**

Set *objOneInfoStore* = *objSession*.**InfoStores**(*index*)

Set *objOneInfoStore* = *objSession*.**InfoStores**(*name*)

objInfoStoresColl

Object. An InfoStores collection object.

objSession

Object. The Session object.

objOneInfoStore

Object. A single InfoStore object.

index

Long. Specifies the number of the message store within the InfoStores collection. Ranges from 1 to the value specified by the InfoStores collection's **Count** property.

name

String. The value of the **Name** property of the InfoStore object to be selected.

Data Type

Object (InfoStore or InfoStores collection)

Remarks

The **InfoStores** property returns a specific message store or 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 message store containing the user's personal messages. The public folders store typically has "Public Folders" in its **Name** property and is always positioned first in the collection, so that its **Index** property always has the value 1.

If your application is running as a Microsoft® Windows NT® service, you cannot access the Microsoft Exchange Public Folders through the normal hierarchy because of a notification conflict. You must use the InfoStore object's **Fields** property to obtain the Microsoft Exchange property PR_IPM_PUBLIC_FOLDERS_ENTRYID, property tag &H66310102. This represents the top-level public folder and allows you to access all other public folders through its **Folders** property. For more information on Windows NT services, see the Win32® Web page *Using MAPI from a Windows NT Service* at <http://www.microsoft.com/win32dev/mapi/mapiserv.htm>.

When you know the unique identifier for a particular InfoStore object, you can obtain it directly with the Session object's **GetInfoStore** method.

Although the **InfoStores** property itself is read-only, the collection it returns can be accessed in the normal manner, and the properties on its member InfoStore objects retain their respective read/write or read-only accessibility.

Example

' from the functions Session_InfoStores and InfoStores_FirstItem


```
Dim objSession as MAPI.Session
Dim objInfoStoresColl as InfoStores
Dim objInfoStore as InfoStore
' 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
collIndex = 1
Set objInfoStore = objInfoStoresColl.Item(collIndex)
' could be objInfoStoresColl(collIndex) since Item is default property
If objInfoStore Is Nothing Then
    MsgBox "Cannot get first InfoStore object"
    Exit Function
Else
    MsgBox "Selected InfoStores item " & collIndex
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)

Group

The **Logoff** method logs off from the MAPI system.

Syntax

objSession.Logoff()

Remarks

The **Logoff** method terminates a MAPI session initiated by the **Logon** method. You can log on to the same Session object again. Attempted access to the Session object before logon results in a return of **CdoE_NOT_INITIALIZED**.

If you have created a ContainerRenderer or ObjectRenderer object while the session was active, you must **Set** it to **Nothing**, or **Set** its **DataSource** property to **Nothing**, before you call the Session object's **Logoff** method. Failure to do so can result in unexpected behavior.

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
```

Logon Method (Session Object)

Group

The **Logon** method logs on to the MAPI system.

Syntax

objSession.**Logon**([*profileName*] [, *profilePassword*] [, *showDialog*] [, *newSession*] [, *parentWindow*] [, *NoMail*] [, *ProfileInfo*])

objSession

Required. The Session object.

profileName

Optional. String. Specifies the profile name to use. To prompt the user to select a profile name, omit *profileName* and set *showDialog* to **True**. The default value is an empty string. The *profileName* parameter is ignored if the *ProfileInfo* parameter is supplied.

profilePassword

Optional. String. Specifies the profile password. To prompt the user to enter a profile password, omit *profilePassword* and set *showDialog* to **True**. The default value is an empty string.

showDialog

Optional. Boolean. If **True**, displays a **Choose Profile** 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 value is **True**. 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 governs the following actions:

| Value | Action |
|--------------|---------------------------------------|
| True | Opens a new MAPI session (default). |
| False | Uses the current shared MAPI session. |

parentWindow

Optional. Long. Specifies the parent window handle for the logon dialog box. A value of zero (the default) specifies that the dialog box should be application-modal. A value of -1 specifies that the currently active window is to be used as the parent window. The *parentWindow* parameter is ignored unless *showDialog* is **True**.

NoMail

Optional. Boolean. Determines whether the session should be registered with the MAPI spooler. This parameter governs the following actions:

| Value | Action |
|--------------|--|
| True | The MAPI spooler is not informed of the session's existence, and no messages can be sent or received except through a tightly coupled message store and transport. |
| False | The session is registered with the MAPI spooler and can send and receive messages through spooling (default). |

ProfileInfo

Optional. String. Contains the server and mailbox names that **Logon** should use to create a new profile for this session. The profile is deleted after logon is completed or terminated. The *ProfileInfo* parameter is only used by applications interfacing with Microsoft® Exchange Server. The *profileName* parameter is ignored if *ProfileInfo* is supplied.

Remarks

The user must log on before your application can use any MAPI object, any other CDO Library object, or even any other method or property of the Session object. An attempt to access any programming element prior to a successful **Logon** results in an **CdoE_NOT_INITIALIZED** error return. The only exception to this rule is the Session object's **SetLocaleIDs** method.

The **Choose Profile** dialog box is always modal, meaning the parent window is disabled while the dialog box 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 box is active. If the *parentWindow* parameter is supplied but is not valid, the call returns **CdoE_INVALID_PARAMETER**.

If your application cannot obtain the handle for the currently active window, for example if it is running in VBScript, you can pass -1 in the *parentWindow* parameter. CDO then retrieves the handle from the Microsoft Windows® **GetActiveWindow** function and uses it as the parent window handle.

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.

The *ProfileInfo* parameter is used to create a temporary profile for the session. CDO generates a random name for the profile. For an authenticated profile, the format of the string is

<server name> & vbLf & <mailbox name>

where the server and mailbox names can be unresolved. Note that the mailbox name is not the messaging user's display name, but rather the alias or account name used internally by the user's organization. For example, "johnd" should be used instead of "John Doe".

For an anonymous profile, the format is

<server distinguished name> & vbLf & vbLf & "anon"

where the distinguished name of the server takes the form

/o=<enterprise>/ou=<site>/cn=Configuration/cn=Servers/cn=<server>

and any text between the **vbLf** characters is ignored. At least the */cn=<server>* entry is required; if it is not specified in the *ProfileInfo* parameter, **Logon** returns **CdoE_INVALID_PARAMETER**.

If you log on with an anonymous profile, the **Name** property of the AddressEntry object returned by the **CurrentUser** property contains "Unknown default".

If both *profileName* and *ProfileInfo* are supplied, *profileName* is ignored and the random profile name is used.

The **Logon** method does not verify the validity of either the server name or the mailbox name in the *ProfileInfo* parameter. You can get a successful return even if you specify one or both of these names incorrectly. In this case the **CurrentUser** property returns the value "Unknown". If you log on using *ProfileInfo*, you should attempt to open the Inbox folder to verify that you can access the message store.

If your application calls the **Logon** method after the user has already successfully logged on to the same session, CDO generates the error **CdoE_LOGON_FAILED**. However, you can create multiple sessions and log on simultaneously each of them.

A session is terminated by the **Logoff** method.

For more information, see [Starting a CDO Session](#).

The following methods can invoke dialog boxes:

- **Details** method (AddressEntry object)
- **Options** and **Send** methods (Message object)
- **Resolve** method (Recipient object)
- **Resolve** method (Recipients collection)

- **AddressBook** and **Logon** methods (Session object)

However, if your application is running as a Microsoft® Windows NT® service, for example from Active Server Pages (ASP) script on a Microsoft® Internet Information Server (IIS), no user interface is allowed.

For more information on Windows NT services, see the Win32® Web page *Using MAPI from a Windows NT Service* at <http://www.microsoft.com/win32dev/mapi/mapiserv.htm>. For more information on running as a service, see "Windows NT Service Client Applications" in the *MAPI Programmer's Reference*.

Example

The first part of this code fragment displays a **Choose Profile** dialog box that prompts the user to enter a profile password. The second part supplies the *profileName* parameter and does not display the dialog box:

```
Dim objSession As MAPI.Session
' (part 1) from the function Session_Logon
Set objSession = CreateObject("MAPI.Session")
If Not objSession Is Nothing Then
    objSession.Logon showDialog:=True
End If

' (part 2) from the function Session_Logon_NoDialog
Function Session_Logon_NoDialog()
On Error GoTo error_actmsg
' 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_actmsg:
If 1273 = Err Then ' VB4.0: If Err.Number = CdoE_LOGON_FAILED 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
```

MAPIOBJECT Property (Session Object)

Group

The **MAPIOBJECT** property returns an **IUnknown** pointer to the Session object. Not available to Microsoft® 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 accessible only by C/C++ programs that deal with **IUnknown** objects. Visual Basic supports the **IDispatch** interface and not **IUnknown**. The **MAPIOBJECT** property is an **IUnknown** object that returns an **IMAPISession** interface in response to **QueryInterface**. For more information, see [Introduction to Automation and How Programmable Objects Work](#). Also see the "COM and ActiveX Object Services" section of the Microsoft Platform SDK.

If your application uses any **MAPIOBJECT** or **RawTable** properties, it must **Release** them all before calling the [Session](#) object's **Logoff** method. Failure to do so can result in unexpected behavior.

The **MAPIOBJECT** property does not correspond to a MAPI property and cannot be rendered into HTML hypertext by the CDO Rendering Library.

Name Property (Session Object)

Group

The **Name** property returns the display name of the profile logged on to this session. Read-only.

Syntax

objSession.Name

The **Name** property is the default property of a Session object, meaning that *objSession* is syntactically equivalent to *objSession.Name* in Microsoft® Visual Basic® code.

Data Type

String

Remarks

To **Name** property contains the current profile's display name. To obtain the messaging user's display name, use the **Name** property of the AddressEntry object returned by the CurrentUser property.

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

Examples

```
' from the function Session_Name
  If objSession Is Nothing Then
    MsgBox "Must log on first: see Session menu"
    Exit Function
  End If
  MsgBox "Profile name for this session = " & objSession.Name
```

OperatingSystem Property (Session Object) Group

The **OperatingSystem** property returns the name and version number of the current operating system. Read-only.

Syntax

objSession.**OperatingSystem**

Data Type

String

Remarks

CDO returns strings in the following formats:

| Operating system | String value |
|----------------------------------|--------------------------------------|
| Microsoft Windows for Workgroups | Microsoft® Windows(TM) <i>N.k</i> |
| Microsoft Windows 95 | Microsoft® Windows 95(TM) <i>N.k</i> |
| Microsoft Windows NT | Microsoft® Windows NT(TM) <i>N.k</i> |

The *N.k* values are replaced with the actual version numbers. Note that Microsoft® Windows® for Workgroups version 3.11 returns the string "Microsoft® Windows(TM) 3.10". This is due to that operating system rather than to CDO.

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)

Group

The **Outbox** property returns a Folder object representing the current user's Outbox folder. Read-only.

Syntax

objSession.Outbox

Data Type

Object (Folder)

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, CDO 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
```

OutOfOffice Property (Session Object)

Group

The **OutOfOffice** property indicates whether the user is currently out-of-office (OOO). Read/write.

Syntax

objSession.**OutOfOffice**

Data Type

Boolean

Remarks

The **OutOfOffice** property is set to **True** when the messaging user wishes to be considered out-of-office and have automatic responses generated to incoming messages. When this property is **True**, CDO signals this condition to the Microsoft® Exchange Client, which then replies to incoming e-mail with the message text contained in the **OutOfOfficeText** property.

For more information on using **OutOfOffice** and **OutOfOfficeText**, see the example in the **OutOfOfficeText** property.

OutOfOfficeText Property (Session Object) Group

The **OutOfOfficeText** property contains the message text for an out-of-office (OOF) response. Read/write.

Syntax

objSession.**OutOfOfficeText**

Data Type

String

Remarks

When the **OutOfOffice** property is set to **True**, the Microsoft® Exchange Client uses the text in the **OutOfOfficeText** property to generate an automatic reply to incoming messages.

Example

This code fragment sets a messaging user to be considered out-of-office and prepares an appropriate response:

```
' Set current user to be out-of-office (OOF)
' assume objSession is a valid Session object
  With objSession
    .OutOfOffice = True
    .OutOfOfficeText = "I'm out of town until next Tuesday"
  End With
  MsgBox "Remember to set .OutOfOffice = False on Tuesday!"
```

SetLocaleIDs Method (Session Object)

Group

The **SetLocaleIDs** method sets identifiers that define a messaging user's locale.

Syntax

objSession.**SetLocaleIDs**(*LocaleID*, *CodePageID*)

objSession

Required. This Session object.

LocaleID

Required. Long. The locale identifier (LCID) to be used for this messaging user.

CodePageID

Required. Long. The code page identifier to be used for this messaging user.

Remarks

A locale is the set of features of a messaging user's environment that are dependent on language, country, culture, and conventions. These features include the character selection, the collating sequence and sort order, and the date, time, and currency formats. The **SetLocaleIDs** method sets identifiers that determine the behavior of locale-sensitive operations.

A locale identifier (LCID) is a 32-bit value containing a 16-bit language identifier and a 4-bit sort identifier. The Microsoft® Windows NT® macros **SORTIDFROMLCID** and **LANGIDFROMLCID** can be used to extract these identifiers from the LCID.

A code page identifier is a long integer specifying the ordered character set to use when displaying text. Information about a code page can be obtained from the Windows NT **GetCPInfo** function.

The **SetLocaleIDs** method does not affect the locale or code page of any rendering object. The RenderingApplication object exposes the **CodePage** property to select the character representation and the **LCID** property to identify the collating sequence, the sort order, and the formats for time, date, and currency representation. Similarly, the ContainerRenderer provides a **CodePage** property and an **LCID** property, and the ObjectRenderer object exposes its own **CodePage** and **LCID**.

If **SetLocaleIDs** is to be called, it must be called before the Session object's **Logon** method is called. This allows the messaging user's profile to be set for the appropriate locale. A call to **SetLocaleIDs** following logon returns **CdoE_CALL_FAILED**.

Note that the **SetLocaleIDs** method is the sole exception to the rule that a call to a session's **Logon** method must precede any other access to that session.

SetLocaleIDs tests the validity of the code page specified by the *CodePageID* parameter before actually setting the locale identifiers. If the code page is not valid, **CdoE_INVALID_ARGUMENT** is returned.

The *LocaleID* and *CodePageID* parameters correspond to the Microsoft Exchange properties PR_LOCALE_ID and PR_CODE_PAGE_ID.

SetOption Method (Session Object)

Group

The **SetOption** method sets a calendar rendering option for the session.

Syntax

objSession.**SetOption**(*OptType*, *OptValue*)

objSession

Required. The Session object.

OptType

Required. String. Selects the option by name.

OptValue

Required. String. Contains the value to set the option to.

Remarks

The calendar rendering options can be inspected with the **GetOption** method. A CDO Rendering application can use their values to set the corresponding properties of a ContainerRenderer object when the rendering application logs on to the session. The container renderer properties are used by the **RenderAppointments**, **RenderDateNavigator**, and **RenderEvents** methods of a CalendarView object to render calendar data. The CDO Rendering ObjectRenderer object can also use the value of the "TimeZone" option.

Note that the rendering objects do not automatically assimilate the values of the Session object's calendar rendering options. You must transfer these values yourself between the Session object and the appropriate properties of the rendering object you are using.

GetOption and **SetOption** return **CdoE_CALL_FAILED** if you call them while you are not logged on to the session. Following a successful logon, **GetOption** returns the option settings that were stored in the messaging user's Inbox with **SetOption**. If no options have ever been stored for the messaging user, the default settings are returned.

The "CalendarStore" option can be "Outlook" or "SchedulePlus", depending on the store underlying the messaging user's calendar folder. This option is useful if you have both types of calendar store and need to select between them. It defaults to "Outlook" if you have neither type or both types. Calendar store information is irrelevant to the container renderer, so "CalendarStore" has no corresponding rendering object property.

If you attempt to set the "CalendarStore" option to anything other than "Outlook" or "SchedulePlus", **SetOption** returns **CdoE_INVALID_PATAMETER**.

The calendar rendering options that can be selected are as follows:

| OptType string value | Data type | Corresponding <u>ContainerRenderer</u> property | Default value |
|-----------------------------|-------------------------|--|----------------------|
| "BusinessDayEndTime" | Variant (vbDate format) | <u>BusinessDayEndTime</u> | 5:00 PM (17:00) |
| "BusinessDayStartTime" | Variant (vbDate format) | <u>BusinessDayStartTime</u> | 9:00 AM (09:00) |
| "CalendarStore" | String | (none) | Current calendar |

| | | | |
|------------------|---------|------------------------------|---|
| "FirstDayOfWeek" | Long | <u>FirstDayOfWeek</u> | store or
"Outlook"
7 (Sunday) |
| "Is24HourClock" | Boolean | <u>Is24HourClock</u> | False |
| "TimeZone" | Long | <u>TimeZone</u> | Current
zone on
Web server |
| "WorkingDays" | Long | <u>BusinessDays</u> | 62
(CdoMonday ...
CdoFriday
) |

The session's options and the container renderer's properties are not automatically kept in synchronization. Changing an option does not cause the corresponding property to be changed, nor vice versa. Your application is responsible for setting all options and properties as appropriate.

Note that the "FirstDayOfWeek" option and the **FirstDayOfWeek** property are compatible with each other in using an enumeration starting with 1 for Monday and ending with 7 for Sunday, but they are not compatible with the **DayOfWeekMask** property of the **RecurrencePattern** object, which uses the mask constants **CdoSunday** through **CdoSaturday**. The "WorkingDays" option, however, is compatible with the mask constants.

Version Property (Session Object)

Group

The **Version** property returns the version number of CDO as a string, for example "1.2.1". Read-only.

Syntax

objSession.Version

Data Type

String

Remarks

The version number for CDO is represented by a string in the form *N.k*, where *N* represents a major version number and *k* 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 ' or Dim objSession As MAPI.Session
Set objSession = CreateObject("MAPI.Session")
' error handling here ...
MsgBox "Welcome to CDO version " _
    & objSession.Version
```

Overview of CDO Rendering

This section is an introduction to the Microsoft® Collaboration Data Objects (CDO) Rendering Library version 1.2.1. It describes rendering and how it is accomplished using the objects in the library. It also provides a short description of Automation and a conceptual overview of the CDO Rendering Library.

The CDO Rendering Library is associated with the CDO Library, and the two are interdependent. CDO Rendering entails displaying CDO objects and collections over the World Wide Web. For more information on CDO, see the CDO Library [Overview](#).

The World Wide Web and HTML

The World Wide Web, or Web, is a distributed information retrieval system that employs multiple protocols on the Internet. The Web operates on a client/server model, where the client is a Web browser and the server is a Web server. When a browser requests a resource from a server, it identifies that resource by means of a formalized address known as a Uniform Resource Locator (URL). The format of a URL is:

protocol://server.network.domain/path/resource

This sample URL uses HTTP (Hypertext Transfer Protocol) to specify the SDK Start Page of the Microsoft® Developer Network Online within Microsoft's home page on the Web:

<http://www.microsoft.com/msdn/sdk/default.htm>

A Web server can respond to an HTTP URL with a Hypertext Markup Language (HTML) document. The hypertext in this document is interpreted by the browser to generate an interactive display on the user's screen. An HTML document is commonly stored in an .HTML or .HTM file.

A Web server can also respond to an HTTP URL with an Active Server Pages (ASP) document, which contains hypertext and ASP script, and is stored in an .ASP file. The script is processed by Microsoft® Internet Information Server (IIS), which generates normal HTML output and sends it to the browser. IIS version 3.0 or later is required for proper handling of .ASP files.

The CDO Rendering Library exposes a set of objects that can be used by IIS to render CDO objects and properties into HTML output. The CDO Rendering objects are described in the remainder of this section. The following table lists these objects in alphabetic order and gives the purpose of each one.

| Object | Purpose |
|-----------------------------|--|
| <u>CalendarView</u> | Specify tabular rendering for a calendar object. |
| <u>Column</u> | Specify rendering for one property in a table view. |
| <u>Columns</u> collection | Provide rendering for every renderable property in a table view. |
| <u>ContainerRenderer</u> | Render a container object. |
| <u>Format</u> | Specify rendering for one property. |
| <u>Formats</u> collection | Provide rendering for every renderable property of an object. |
| <u>ObjectRenderer</u> | Render selected properties of a CDO object. |
| <u>Pattern</u> | Specify rendering for certain values of a property. |
| <u>Patterns</u> collection | Provide rendering for all values of a property. |
| <u>RenderingApplication</u> | Create rendering objects. |
| <u>TableView</u> | Specify tabular rendering for a container object. |
| <u>Views</u> collection | Provide a selection of views for rendering a container object. |

HTML Rendering

The purpose of CDO Rendering is to generate displayable output from CDO objects and properties. The output is sent in Hypertext Markup Language (HTML) to a Web browser and is generated from objects and properties referenced in an .ASP file invoked by the browser. This process is known as HTML rendering.

An .ASP file is a specialized HTML file containing hypertext, client-side script, and Active Server Pages (ASP) script for a Web page. The script can instantiate objects, call their methods, manipulate their properties, and produce results relating to the Web page. Client-side script is decoded and run by the browser itself. ASP script is decoded and run by Microsoft® Internet Information Server (IIS) to supply Web pages to the browser.

Both client-side and ASP script can be written in any scripting language, such as Microsoft® Visual Basic® Scripting Edition (VBScript), JScript™, or JavaScript. Script can be inserted anywhere in an HTML document, that is, anywhere between the <HTML> and </HTML> tags.

Client-side script is delineated by the <SCRIPT> and </SCRIPT> tags. The scripting language's compiler or interpreter is invoked by the browser. Client-side script typically responds to input from the browser's user. For example, it can create and send an e-mail message when the user clicks a button. For more information on client-side script, see [Web Page Support](#).

ASP script can be delineated by either the <% and %> tags or the <SCRIPT RUNAT=SERVER> and </SCRIPT> tags. The compiler or interpreter is invoked by an IIS component before the document is sent to the browser. ASP script uses the CDO Rendering Library to perform HTML rendering. It can, for example, prepare a displayable rendition of a folder. ASP script is not sent to the browser, but instead is used to generate hypertext that is sent in its place.

ASP script can be embedded within client-side script. This is useful, for example, when an exception condition occurring at the server needs to be displayed at the browser. You can only use one level of embedding, and you must delineate your embedded ASP script with the <% and %> tags.

If the browser invokes a Uniform Resource Locator (URL) specifying an .HTM file, IIS sends the file to the browser without modification, just as any Web server does. If, however, the URL specifies an .ASP file, IIS first searches it for any ASP script. If there is none, the file is treated as a normal .HTM file and sent directly to the browser. If IIS does find ASP script, it calls the appropriate compiler or interpreter to execute the script and generate HTML hypertext. IIS then replaces the script with the hypertext and sends the result to the browser as if it were an .HTM file. Note that when ASP script is executed the server does not retain any persistent file containing the exact stream sent to the browser.

Writing a Rendering Application

To render CDO objects and properties into HTML hypertext, you instantiate a rendering object such as a [ContainerRenderer](#) or [ObjectRenderer](#) object. These are top-level objects, which can be created directly by your code without deriving them from any other object. For more information, see [CDO Rendering Objects](#).

The recommended approach, however, is to begin by instantiating a [RenderingApplication](#) object, from which you can create a family of related rendering objects. The [RenderingApplication](#) object allows you to set options which are inherited by every rendering object you create from it. You can choose the verbosity of event logging in each of several categories using the [LoggingLevel](#) property, set the code page for character representation with the [CodePage](#) property, and prepare specific rendering information through the [Formats](#) property. These options act as global presets for your entire rendering application. The individual rendering objects can change their inherited copies of the code page and the formats if appropriate.

The next step is to use the [CreateRenderer](#) method to instantiate a specific rendering object, such as a [ContainerRenderer](#) or [ObjectRenderer](#) object. The rendering object is used to generate HTML hypertext from a CDO object. You create a container renderer if you want to render the contents of a container object, such as an address book container or a folder. You create an object renderer if you only want to render one or more selected properties of an object, such as when a message was sent and received.

After you create the rendering object, you set its [DataSource](#) property to specify the object or collection you want to render. The container renderer's [DataSource](#) property accepts an [AddressEntries](#), [Folders](#), [Messages](#), or [Recipients](#) collection. The object renderer's [DataSource](#) property accepts an [AddressEntry](#), [AppointmentItem](#), [Attachment](#), [Folder](#), [MeetingItem](#), or [Message](#) object.

Every rendering object has a child [Formats](#) collection, which is originally inherited from the parent [RenderingApplication](#) object but can be modified as needed. You can add a new [Format](#) object with the collection's [Add](#) method and delete a format with the [Format](#) object's [Delete](#) method. You should make changes to the [Formats](#) collection before setting the [DataSource](#) property; otherwise they are not reflected in the rendering of the new data source.

A [Format](#) object provides information for rendering one property of the object being rendered, or one property of the items in the container object being rendered. Each format controls exactly one property, and each property to be rendered must be represented by exactly one format. You specify the property in the format's [Property](#) property. The rendering information is contained in the format's [Patterns](#) collection, which is accessed with the format's [Patterns](#) property.

Each [Pattern](#) object furnishes information for rendering a particular set of values of the property. You specify the value set in the pattern's [Value](#) property and a rendering source for that value set in the [RenderUsing](#) property. The rendering source is a string containing HTML hypertext and substitution tokens. When the property's value matches a pattern's value set, that pattern's rendering source is used to render the property.

Rendering Container Objects

A container object can be a CDO [AddressEntries](#), [Folders](#), [Messages](#), or [Recipients](#) collection. To render any container object and its contents, you use a [ContainerRenderer](#) object.

When you render a container object you apply a view to it. After you have specified the container object in the container renderer's [DataSource](#) property, a [Views](#) collection is available through the container renderer's [Views](#) property. From this collection you select the appropriate [CalendarView](#) or [TableView](#) object and set the [CurrentView](#) property to that object. If you do not make a selection, the current view remains set to the default view, which is the first view in the collection unless specified otherwise by the underlying store.

A calendar view or table view is normally defined externally to your rendering application. For a folder there are three types of predefined table views: *common views*, *folder views*, and *personal views*. Common views are defined globally for all folders and all messaging users. A set of folder views can be defined for each individual folder, and a set of personal views for each individual messaging user. An address book container has no common, folder, or personal views predefined. The type of a calendar view is available in its [Source](#) property, and that of a table view in its [Source](#) property.

Modifications you make to predefined views do not persist in storage, but they remain in effect for your application until you change the source of the view. Modifications to folder views last until the [DataSource](#) property is changed, and modifications to common and personal views last until the [ContainerRenderer](#) object is released. Simply changing the [CurrentView](#) property does not nullify your modifications.

In addition to the predefined views for the container object you are rendering, you can create new *custom views* with the collection's [Add](#) method. This is the only type of view you can use when rendering an address book container. A custom view is nonpersistent and cannot be saved. However, it is still available when the [DataSource](#) property is changed. You cannot delete any views from the Views collection.

A [CalendarView](#) or [TableView](#) object specifies the overall rendering of a container object through its [Columns](#) property, which returns a child [Columns](#) collection for the table view. The order of the columns in the collection determines their display order. You can use the collection's [Add](#) method to add new columns, which cease to exist when the collection is released, for example when the [DataSource](#) property is changed. Columns cannot be deleted from their collection.

Each [Column](#) object corresponds to one property of the items in the container object being rendered. The Column object specifies the renderable property in its [Property](#) property and a rendering source for that property in the [RenderUsing](#) property. As with a pattern, the rendering source is a string containing HTML hypertext and substitution tokens. The Column object also specifies the relative width of the column in its [Width](#) property and control flags for the rendering in its [Flags](#) property.

The modifications you can make to a column consist of changing its [Flags](#), [RenderUsing](#), and [Width](#) properties. The modifications you can make to an existing calendar view or table view consist of adding and changing columns in its child Columns collection. Column modifications remain in effect until the Columns collection is released, which is usually when the [DataSource](#) property is changed. Calendar view and table view modifications remain in effect until the source of the view is changed, namely the [DataSource](#) property for custom and folder views, or the [ContainerRenderer](#) object for common and personal views.

Rendering a Calendar

A calendar is rendered under control of a ContainerRenderer object and a CalendarView object. The calendar view operates on a Messages collection in a calendar folder. The container renderer specifies the Messages collection in its DataSource property and the calendar view to be applied in its CurrentView property.

A calendar folder is usually obtained from the Session object's GetDefaultFolder method, with CdoDefaultFolderCalendar in the *ObjectType* parameter. The calendar folder is expected to contain only AppointmentItem objects in its Messages collection. Successful calendar rendering depends upon each appointment's StartTime and EndTime properties being set properly.

A calendar view is normally generated externally to a CDO application. The predefined common views are identified by "Daily" and "Weekly" in the Name property. You can also create a nonpersistent custom view with the Add method of the Views collection. A calendar view created in this way ceases to exist when the collection is released.

A calendar view bases its HTML output on the following display units:

- The *time unit*, which can be a day or a week, and around which the view is built;
- The *time span*, which is the total amount of calendar time displayed in a single HTML page;
- The *time slot*, which is the smallest subdivision within each day in the view.

The time unit is specified by the view's Mode property. The time span is expressed as a multiple of time units in the view's NumberOfUnits property. The size of the time slots is controlled by the view's Interval property. Time slots appear as a grid when the Mode property is set to CdoModeCalendarDaily.

The calendar view, which is subclassed from the TableView object, is structured using rows and columns. Each column of the view represents one day, and is rendered as a one-column-wide HTML table. Each cell in the table, at the intersection of a row and a column, displays one time slot.

AppointmentItem objects with their Duration property equal to an exact multiple of 24 hours are considered events; otherwise they are appointments. You can render appointments using the calendar view's RenderAppointments method and events with the RenderEvents method. Events also have their AllDayEvent property set to **True**.

You can allow the user to determine the starting date for rendering by calling the RenderDateNavigator method, which prepares HTML output for one or more months in traditional calendar format. The HTML page interacts with the user, who can select a date from a displayed month or move forward and backward through the months as necessary.

You can set properties on the calendar view to control the rendering of different types of cells. The BusyCell property specifies rendering for time slots that are busy, that is, those that are within the duration of an accepted appointment. The FreeBusinessCell and FreeNonBusinessCell properties determine the rendering of free time slots that are within or outside of business hours, respectively. Business hours are determined by the ContainerRenderer object's BusinessDayStartTime and BusinessDayEndTime properties, and business days by the BusinessDays property.

In a daily view, the RenderAppointments method renders a narrow vertical column, called the *indicator bar*, between the time slot column on the left and the appointments table on the right. The indicator bar is intended to make it easier for the user to envision the state of the calendar with respect to busy and free times. It can be rendered differently for different degrees of commitment of a time slot. You can set the various renderings of the indicator bar with the calendar view's FreeIndicator, TentativeIndicator, BusyIndicator, and OOIndicator properties.

Rendering Object Usage

The following table summarizes the CDO Rendering objects and their principal functions, in their usual order of exploitation.

| Object | Principal function |
|-----------------------------|---|
| <u>RenderingApplication</u> | Set global rendering options and create container renderers and object renderers using the CreateRenderer method. |
| <u>ContainerRenderer</u> | Specify a container object in the DataSource property and render it using the Render , RenderHeading , and RenderProperty methods. |
| <u>ObjectRenderer</u> | Specify a CDO object in the DataSource property and render selected properties of that object using the RenderProperty method. |
| <u>Formats</u> collection | Provide rendering control for every renderable property of the object being rendered. |
| <u>Format</u> | Specify one property to be rendered in the Property property and provide rendering information in the patterns accessed with the Patterns property. |
| <u>Patterns</u> collection | Provide rendering information for all possible values of the property being rendered under the control of a format. |
| <u>Pattern</u> | Specify a particular value set of the property being rendered in the Value property and provide rendering information for that value set in the RenderUsing property. |
| <u>Views</u> collection | Provide a selection of views for rendering a container object, from which a container renderer can choose using its CurrentView property. |
| <u>TableView</u> | Provide tabular rendering information for every renderable property of a container object being rendered, which can be accessed with the Columns property. |
| <u>CalendarView</u> | Provide tabular rendering information for every renderable property of a calendar object being rendered, which can be accessed with the Columns property. |
| <u>Columns</u> collection | Provide rendering information for every column to be rendered in a table view, including the display order of the columns. |
| <u>Column</u> | Specify one property for a table view in the Property property and provide column rendering information for that property in the RenderUsing property. |

All URL strings and .ASP file names must be entirely in 7-bit ASCII representation. This provides maximum code page commonality.

To make use of the CDO Rendering Library, make sure it is referenced by the automation controller you are using. With Microsoft® Visual Basic®, for example, run the **References** command and select

the check box for **Collaboration Data Objects Rendering Library 1.2.1**.

CDO Rendering Library Design

The CDO Rendering Library is designed for flexibility and performance. It implements HTML rendering of the CDO objects most used by client applications. The CDO Rendering Library is not designed for development of service providers.

The CDO Rendering Library is based on the capabilities provided by Automation. It allows you to create instances of programmable rendering *objects* that you can reference with automation controllers. An *automation controller* is a tool that supports Automation, such as Microsoft® 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. Throughout this document, Visual Basic is used as a concrete example of an automation controller, but the statements about Visual Basic apply to all such tools.

CDO Rendering Objects

Objects in the CDO Rendering Library can be classified as *top-level objects*, *child objects*, and *collections*. A top-level object is one that can be created directly by your code, without having to derive it from any other object. A child object is one that must be derived from another object, for example by an **Add** method. A collection is a group of objects of the same type.

Currently, the top-level CDO Rendering objects are the [RenderingApplication](#), [ContainerRenderer](#), and [ObjectRenderer](#) objects. Other objects are accessible only through these top-level objects.

You can create a RenderingApplication object either through early binding:

```
Dim objRendApp As RenderingApplication
Set objRendApp = CreateObject ("AMHTML.Application")
```

or through late binding:

```
Dim objRendApp As Object
Set objRendApp = CreateObject ("AMHTML.Application")
```

and then later on use the [CreateRenderer](#) method to create specific rendering objects.

C/C++ programmers use globally unique identifiers (GUIDs) for these objects, defined in the type library for the CDO Rendering Library. The following table lists the GUIDs for the top-level objects accessible to C/C++ programmers. Note the close relationship; only the fourth of the 16 bytes differs among the GUIDs.

| CDO Rendering Library object | GUID |
|--------------------------------------|--|
| RenderingApplication | {BC00F701-31AC-11D0-B5F1-00AA00BF3382} |
| ContainerRenderer | {BC00F703-31AC-11D0-B5F1-00AA00BF3382} |
| ObjectRenderer | {BC00F702-31AC-11D0-B5F1-00AA00BF3382} |

All CDO Rendering Library objects can be considered as relative to a [RenderingApplication](#) object. A rendering application's immediate child objects are the [ContainerRenderer](#) object and the [ObjectRenderer](#) object. These have their own child objects, which in turn have child objects, and so on. See the [Rendering Object Model](#) diagram for the logical hierarchy of the CDO Rendering Library.

The object hierarchy is important because it determines the correct syntax to use in your Microsoft® 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,

```
objContRend.Formats.Item(2)
```

refers to the second Format object in the Formats collection of the current ContainerRenderer object.

CDO Rendering Collections

The CDO Rendering Library supports the following collections:

Columns

Formats

Patterns

Views

The Views collection can hold view objects of different classes. The currently supported view classes are represented by the CalendarView and TableView objects.

CDO Rendering Library collections all have a **Count** property, which always contains the current number of member objects. Every collection also has an **Item** property, which can be used to select any arbitrary member of the collection. Each object in a collection has an **Index** property, assigned by the CDO Rendering Library. The **Index** value for the first member object is 1. An object's **Index** property can be used as an attribute of the collection's **Item** property to reselect that object later.

Index properties are valid only during the current access to the collection and can change as your application adds and deletes objects. For example, in a Formats collection with three Format objects, the first format is referred to as Formats.Item(1), the second as Formats.Item(2), and the third as Formats.Item(3). If your application deletes the second format, the third format becomes the second and Formats.Item(3) has the value **Nothing**. The **Count** property is always equal to the highest **Index** currently in the collection. Note that the **Count** is refreshed when you repopulate the collection.

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

Rendering Messaging Objects and Collections

Messaging objects and object collections are rendered in order to assemble HTML pages. This is done when server-side Active Server Pages scripts generate HTML into the current HTTP output using the Active Server Pages Response object.

The properties of collections of objects are rendered differently than are properties of individual objects. The following sections illustrate typical steps you would take in each case and the various rendering options available.

Note Some of the concepts described in this section are illustrated with code examples from Microsoft Outlook Web Access. To gain access to the source .asp files of Outlook Web Access, install it from the Microsoft Exchange Server installation CD by following the steps in Installing Microsoft Outlook Web Access.

Rendering Individual Object Properties

The ObjectRenderer object is used to render one or more properties, rather than an entire object. The following procedure shows the steps you would take to render a property on an object.

Group

To render an object property

1. Open the MAPI object, such as a message or a user's Inbox. For example, to open the Inbox belonging to logged-in user, use objSession.Inbox.
2. If it does not already exist, create the ObjectRenderer object with a call such as objRenderApp.CreateRenderer (class). For more information, see About Renderer Objects.
3. Set the **DataSource** property to the object that contains properties to be rendered with a command such as Set objObjectRenderer.DataSource = objMessage. For more information, see Setting the Data Source.
4. Optionally, depending on the property type of the property to be rendered, set Formats.
5. Render the property. This call specifies what property of the object to render. For example, to render the subject of the message, you would have set the data source to be the message, and call the **RenderProperty** method to render the subject, specifying the MAPI property. You have the choice of rendering either directly to the screen or to a string.

Rendering Collections

The ContainerRenderer object is used to render one or more properties from a collection of MAPI objects, as shown in the following procedure.

Group

To render a collection

1. Open the collection within the MAPI object, such as the Inbox.Messages collection or the Inbox.Folders collection. For example, to open the Messages collection of the logged-in user's Inbox, use objSession.Inbox.Messages.
2. If it does not already exist, create the ContainerRenderer object with a call such as objRenderApp.CreateRenderer Method (class). For more information, see About Renderer Objects.
3. Set the DataSource to the collection to be rendered with a command such as Set objCR.DataSource = Inbox. For more information, see Setting the Data Source.
4. Optionally, to create active links, set the LinkPattern property. For more information, see Named Formats.
5. Add any additional Formats needed to render your collection. For example, to render the contents table of an Inbox, you will probably need to include formats for the MAPI properties PR_IMPORTANCE, PR_MESSAGE_CLASS, and PR_HASATTACH.
6. Render the object. For collections, you will generally render directly to the screen.

Creating the RenderingApplication Object

You can usually assume that a RenderingApplication object will at some point be useful for your application. Only rarely would it not; for example, you may want to view mail but your application does not deal exclusively with viewing mail; in this case you can create an ObjectRenderer or ContainerRenderer object when needed, using the Server.CreateObject method instead of using the RenderingApplication object. See [About Standalone Renderer Objects](#).

But even in those rare cases, you may still choose to start by creating a RenderingApplication object. This is because it is faster to create child rendering objects from a RenderingApplication object than by using the Server.CreateObject method.

Therefore, because the **Application_OnStart** procedure in the global.asa file is automatically called by [IIS](#), it is efficient to create the RenderingApplication object by placing the following call in that procedure.

```
CreateObject ("AMHTML.RenderingApp "virtroot" "classpath" ")
```

This call creates two other objects as well, the [virtroot and classpath format objects](#). There is only one RenderingApplication object per [virtual root](#), and therefore only one per application.

Extending the Scope of the RenderingApplication Object

When you create the RenderingApplication object with the Server.CreateObject method in **Application_OnStart**, you assign it to a local variable, which limits its lifetime to that of the **Application_OnStart** function.

Because of this limitation, it is a good idea to store the RenderingApplication object in the Active Server Pages Application object. The Active Server Pages application is global, so storing the RenderingApplication object expands its scope to let it be accessed by other scripts, and extends its lifetime to that of the application. To do this, apply a name to the RenderingApplication object and save it into a table for later reference, using a call like the following:

```
Set Application("RenderingApplication") = objRenderApp
```

You can release the RenderingApplication object by setting its value in the Active Server Pages Application object to Nothing:

```
Set Application("RenderingApplication") = Nothing
```


Configuring the RenderingApplication Object

After creating the RenderingApplication object, you need to configure it by setting its properties. Properties are set using information from these sources:

- Microsoft® Windows NT® registry entries
- Microsoft® Exchange directory store

You can configure the RenderingApplication object by calling the **LoadConfiguration** method on the RenderingApplication object. Calling **LoadConfiguration** retrieves information about the Microsoft Exchange Server from the registry and writes it to a table in the RenderingApplication object. In this call, set the *source* parameter to 1 (AMHTML_Config_Registry) to load information from the key associated with the service.

For Microsoft Outlook Web Access, this key is:

HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\MSExchangeWeb\Parameters

This registry information was originally entered by the administrator who set up Microsoft® Outlook™ Web Access during the installation of Microsoft Exchange Server.

In a separate call, set the *source* parameter to 2 (AMHTML_Config_DS) to load information from the Microsoft Exchange directory service, such as whether HTTP is enabled and what public folders have been published. This call can be found in the **Session_OnStart** function in global.asa:

```
objRenderApp.LoadConfiguration 2, ""
```

Retrieving Configuration Information

Everything that you store using **LoadConfiguration** can later be retrieved through the **ConfigParameter** property of the RenderingApplication object because all this information, whether originally from the registry or from the Microsoft Exchange directory service, is mapped into the same table. You use **ConfigParameter** by passing a single parameter, called *parameter*, to specify the configuration value you want to retrieve. For example,

```
objRendApp.ConfigParameter("Published Public Folders")
```

returns the published public folder shortcuts. Because this information is stored in the registry by name, you can store additional application-specific information in the registry key (with **LoadConfiguration**) and retrieve it later when you need it, using the RenderingApplication object. For example, the following call retrieves the value for the Microsoft Exchange Server saved previously in a registry entry called Server:

```
bstrExchServer = objRenderApp.ConfigParameter("Server")
```

Setting virtroot and classpath

When the RenderingApplication object is created, it contains a [Formats](#) collection object as well as two named formats: the virtroot object and the classpath object. To configure these objects, you set the [VirtualRoot](#) property (to set the virtroot format) and [FormsRoot](#) property (to set the classpath format) as a part of RenderingApplication object configuration. It is good to make these settings right after the first [LoadConfiguration](#) command.

As formats in the formats collection of the RenderingApplication object, the virtroot object and the classpath object are global to the application.

The virtroot Format Object

The virtroot format functions like a symbol in a table in that it lets you provide a link that will resolve to the correct location of the current Web application. You can use virtroot as the substitution token %virtroot% in substitution patterns. For more information, see [VirtualRoot Property \(RenderingApplication Object\)](#).

In [Microsoft Outlook Web Access](#), the [VirtualRoot](#) property is set during configuration of the RenderingApplication object, in the [Application_OnStart](#) procedure in global.asa:

```
bstrVirtRoot = "/exchange"  
objRenderApp.VirtualRoot = bstrVirtRoot
```

The classpath Format Object

The classpath format is used to create links to forms. If you use the classpath format without having set the [FormsRoot](#) property of the RenderingApplication object, classpath functions as a substitution token, returning the message class of the current object.

The classpath format object obtains a new value when you set the [FormsRoot](#) property. In Microsoft Outlook Web Access, this property is set during configuration of the RenderingApplication object, in the [Application_OnStart](#) procedure in global.asa:

```
bstrVirtRoot = "/exchange"  
objRenderApp.FormsRoot = Server.MapPath(bstrVirtRoot) & "\\usa\forms"
```

In this example, the Server.MapPath method expands the string bstrVirtRoot to the [virtual root](#) of /exchange, namely c:\exchsrvr\webdata. This [FormsRoot](#) call also performs the following actions:

- The objRenderApp.FormsRoot property is set to a value such as c:\exchsrvr\webdata\usa\forms.
- The directory tree starting with the specified physical directory (...\\usa\forms) is searched for installed forms—that is, directories containing [ASP Files](#).
- A [pattern](#) for each form found in this directory tree is added to the classpath format object.

Optionally, to store information about additional forms directories, you can add more patterns to the classpath format object. Because the classpath contains only the beginning of the path to the forms directories, it is best to place all forms under the same virtual root, such as /exchange.

For more information, see [RenderUsing Property \(Column Object\)](#) and [FormsRoot Property \(RenderingApplication Object\)](#).

About Renderer Objects

There are two kinds of renderer objects (or "renderers"): the `ObjectRenderer` object and the `ContainerRenderer` object. Using them in conjunction with [Format objects](#) provides the flexibility of displaying information in an interpreted manner.

The rendering of each property of an object is controlled by the various [formats](#) contained in the renderer object. By creating a renderer in advance and presetting its properties, you can prepare for rendering an object that later will become or may become the data source. For more information, see [When to Create Renderer Objects](#).

In general, after creating the renderer and setting its formats and data source, you call the **Render** method. When you call the rendering object's **Render** method, it generates [HTML](#) in a stream that makes up a page about to be sent to the browser.

When to Create Renderer Objects

Because renderer objects can be stored, you can choose the best time to create them:

- Create the renderer object when needed and discard it afterward.
- Create it beforehand and store it until needed—perhaps reusing it several times.

In fact, it is usually most efficient to create a renderer object before opening the object that it will render. Microsoft® Outlook™ Web Access, for example, creates rendering objects for mailboxes before needing them and stores them in the Active Server Pages Session object. Web Access uses only two rendering objects: one `ObjectRenderer` and one `ContainerRenderer`. This means that all its rendering is done through calls to methods on one of these two objects.

Renderer objects were designed to be reused. Stored renderers retain the property values they held for previously rendered objects until these properties are given new values.

Note Do not confuse the `RenderProperty` method of the `ContainerRenderer` object with that object's `Render` method. The `Render` method renders rows of a folder or address book collection. The `RenderProperty` method renders the designated property of the parent of the object specified by the `DataSource` property. It renders the property in place, either to the Active Server Pages Response object or to a `bstr` string which will then be displayed. Note that `bstr` (`strHTML`) is a Pascal-like string construct used by Microsoft® Visual Basic®; it is a NULL-terminated, Unicode, wide-character string used by `IDispatch`.

About Standalone Renderer Objects

The ContainerRenderer and ObjectRenderer objects are known as top-level objects because they can be created directly by your code, without having to be derived from another object. When created directly, such an object is known as a standalone renderer object because it has no parent link—a pointer to the object it was created by or added to.

Because a standalone renderer has no parent link to the RenderingApplication object, it does not have the support methods of the RenderingApplication object. Nor is there a global format collection, so the virtroot and classpath objects would not exist.

To create a ContainerRenderer or ObjectRenderer standalone object, use the Active Server Pages syntax `Server.CreateObject`, naming the object `AMHTML.ContainerRenderer` or `AMHTML.ObjectRenderer`.

In general, standalone renderer objects are best used for single operations, such as rendering certain properties of the current object. When you use standalone renderer objects, you generally do not intend to read more data from the object or access it in other ways.

Renderers need not be standalone. That is, you can choose to derive objects of these types from a RenderingApplication object, as shown in the following section.

Web Access Example: Creating Renderer Objects

In Microsoft Outlook Web Access, when a ContainerRenderer or ObjectRenderer object is required—for example, to render the Web user's Inbox—and is not found in the Active Server Pages Session object, it is created using the existing RenderingApplication object with a call such as:

```
objRenderApp.CreateRenderer ( class )
```

where *class* is the CDO Rendering class enumeration, in this case AMHTML_Class_ObjectRenderer.

Using CheckSession

The **CheckSession** function is a Microsoft® Outlook™ Web Access library function that can be called to determine whether the Active Server Pages session is still valid.

If a Web user times out and then tries to refresh the current page, **CheckSession** determines that the user has indeed timed out—because the Active Server Pages Session object no longer exists. It then requests that the user log in again, providing a dialog box for that purpose (in Web Access, it redirects them to logon.asp). When the Web user logs in again, new MAPI and Active Server Pages sessions are started.

If the Web user logged in anonymously, has timed out, and now refreshes a page, **CheckSession** logs the person in transparently—that is, without requiring user information. This will also be an anonymous login.

It is good to use **CheckSession** on every script page.

Note The **CheckSession** function is defined in the Outlook Web Access file session.inc. To make functions such as **CheckSession** available for use, you must first include this source file, which you can do in your .ASP file with the following command:

```
<!--#include file="../lib/session.inc"-->
```


Setting the Data Source

To display text or images to represent data in Microsoft® Exchange information stores, you render the MAPI properties that hold that data. Using CDO, you can render properties on individual MAPI objects or properties on entire collections of objects. To determine what is rendered, set the **DataSource** property on the renderer object.

About Collections and Individual Objects

You must set the correct kind of data source—an object or a collection—when rendering an individual object or a collection.

To render a collection, use a ContainerRenderer object. A valid data source for a ContainerRenderer would be a Folders collection or a Messages collection. Collection objects can be CDO AddressEntries, Folders, Messages, or Recipients collections.

In contrast, the ObjectRenderer object renders properties on a single MAPI object. It can take any of the following objects as a valid **DataSource**: AddressEntry, Attachment, Folder, InfoStore, Message, and Session.

Because of similarly named objects, take care when setting the **DataSource**. You can pass a folder object as the **DataSource** on the ObjectRenderer object but if you pass it as the **DataSource** on a ContainerRenderer object, the call will fail. However, you can pass either Folder.Folders or Folder.Messages to the ContainerRenderer object because they are collection objects.

Valid Data Sources

Only the following objects can be used as data sources in CDO rendering:

- An individual object (used with an ObjectRenderer object) that has the **MAPIOBJECT** property.
- A collection object (used with a ContainerRenderer object) that has the **RawTable** property.

For example, the InfoStores collection object cannot be used as a data source because it has neither a **MAPIOBJECT** property nor a **RawTable** property. But an InfoStore object (a child object of the InfoStores collection), can be used as a data source for an ObjectRenderer object because it does have a **MAPIOBJECT** property.

The **MAPIOBJECT** property returns an **IUnknown** pointer to the underlying MAPI (COM) object. The **RawTable** property returns a pointer to the **IMAPITable** object.

Web Access Example: Setting a Data Source

In [Microsoft® Outlook™ Web Access](#), the data source is set after the creation of the object that will render the data. For example, a ContainerRenderer object is created and saved first (see [When to Create Renderer Objects](#)). Then, as the Web user navigates through the folder hierarchy, Web Access script sets the **DataSource** property.

For example, the following line sets a MAPI folder as a data source. This is valid because the Folder object contains a **MAPIOBJECT** property.

```
Set objObjectRenderer.DataSource = objFolder
```

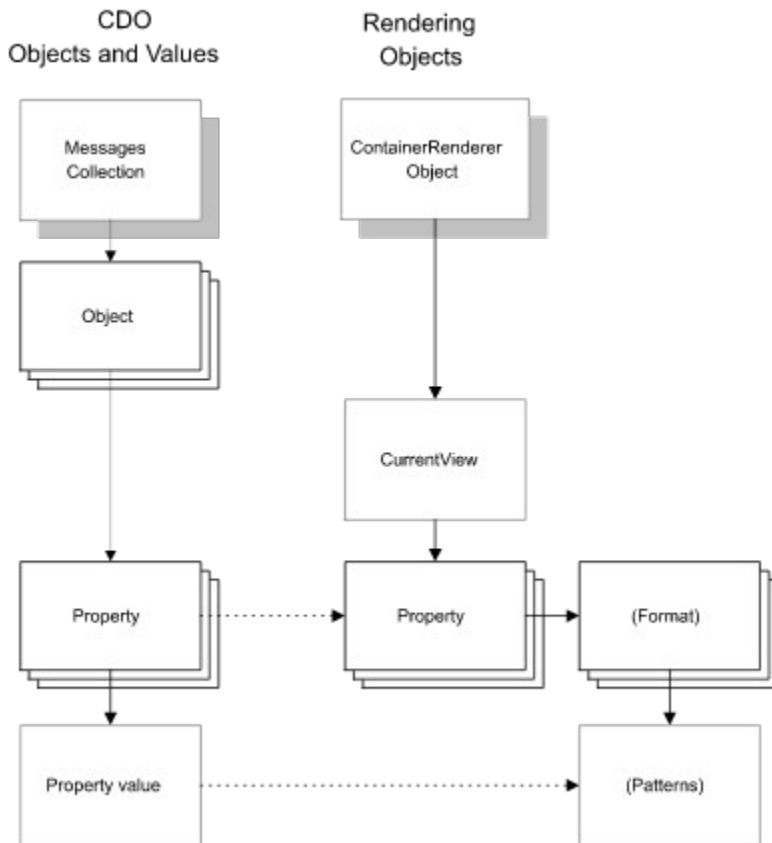
In this call, *objObjectRenderer* is the CDO Rendering object and *objFolder* is the MAPI folder object obtained through CDO calls. This call adds a reference to the MAPI object (in a manner similar to an **AddRef** call) while setting it as the current **DataSource**.

Using Views

A view is a collection of columns. To render a collection, you apply a view to it, which defines which properties of the collection object are rendered as columns on your Web page. After you have specified the collection in the **DataSource** property of the ContainerRenderer object, a views collection becomes available through the **Views** property of the ContainerRenderer. For more information, see Rendering Container Objects.

The following diagram shows how the view and other objects relate in the rendering of collections. A view contains a set of columns to be rendered, each of which represents an object's property. Each column can render a property according to a specific format, which in turn consists of patterns. And patterns map to the various values that a messaging object's property can have.

Object Relationships for Collection Rendering



The column object controls rendering of the property in one of these ways:

1. A named format is referenced in the RenderUsing string of the column.
2. A property-only format is defined, and its **Property** property corresponds to the **Property** property of the column object.
3. No format is defined, and the property is rendered by type, as described in the table Default Output Styles of Supported Property Types in the section About Patterns.

For more information, see About Format Objects and Using Named Properties.

About Format Objects

The format object of the CDO Rendering Library contains specific information that controls how a messaging property of any type is to be rendered. In other words, format objects provide a way to interpret information as you render it.

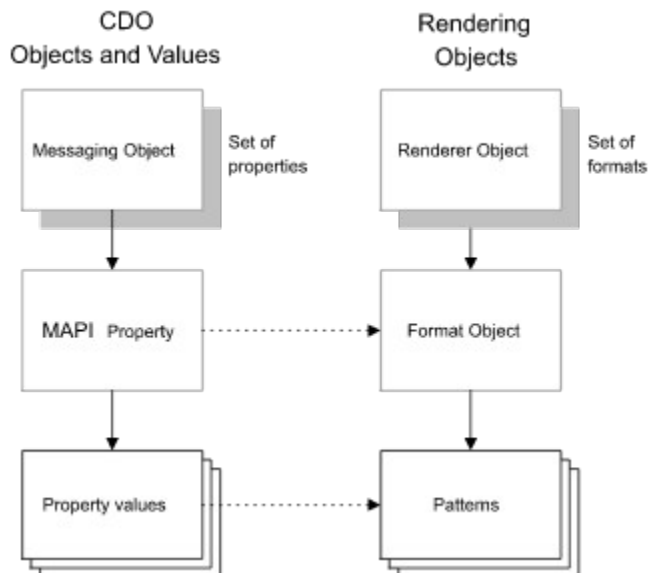
There is a 1:1 relationship between format object and property to be rendered; each format controls exactly one property, and each property to be rendered must be represented by exactly one format. Rendering information is contained in the format's Patterns collection, which is accessed with the format's **Patterns** property.

You need not always render properties using formats. If you used a ContainerRenderer object without adding formats, it would render MAPI data according to certain default rules. For example, it would convert number values into strings and it would convert multivalued string Unicode arrays into a single line of semicolon-separated text. It would also use its own rules for rendering date and time values. Using a format object lets you make exceptions to these default rules by having the renderer display each property value exactly as you want.

There are two mutually exclusive types of formats, whose uses are explained in [Property-Only Formats](#) and [Named Formats](#).

The following diagram shows how formats and other objects relate in the rendering of object properties. A format contains a set of patterns, each of which tells exactly how to render a specific property value of the MAPI property being rendered.





Object Relationships for Property Rendering



Example: Results of Using Formats

The following table shows the results of using formats for rendering a typical set of message properties. These seven properties constitute what is commonly known as the Normal view (of a message collection). If you do not set formats, the properties are rendered as shown in the second column, but if you do set formats, you will see the values or images in the third column.

Effect of Using Formats to Render Properties

| MAPI property | Without a format | With a format |
|---------------------------|-------------------------------------|---|
| PR_IMPORTANCE | 0, 1, or 2 (long integer) | image ( , nothing, or ) |
| PR_MESSAGE_CLASS | IPM.Note | image ( , or other item specific to the message class) |
| PR_HASATTACH | 0 (or 1) | image () |
| PR_SENT_REPRESENTING_NAME | (sender, as a string) | (sender, as a string) |
| PR_SUBJECT | (subject, as a string) | (subject, as a string) |
| PR_RECEIPT_TIME | (date received, in SYSTIME format) | (date received, as a string) |
| PR_MESSAGE_SIZE | 1024 (long integer value, in bytes) | 1 KB (converted to kilobytes) |

In the preceding table, the MAPI PR_HASATTACH property, which contains a 1 or a 0, indicates whether an item has an attachment or not. The meaning of this property is better conveyed with text ("Has an Attachment") or a GIF image of a paper clip, which you can display using an HTML image tag. In this example the image is a pattern that matches the value of 1 for the PR_HASATTACH property.

Property-Only Formats

A property-only format is distinguished by having an empty name property.

The only use of a property-only format is to render the columns of a view. Each column's value is rendered according to the patterns of the format object whose **Property** property has been set to the **Property** property of the column object. This is the most common way of linking a format to a column in a view. For more information, see [Using Views](#).

Using Named Properties

User-defined named properties may also be specified in place of a predefined property value for the **Property** property of a format or column object.

You can identify the property with either a string name (such as "NAMED_PROP") or a numeric identifier (such as 0x7E00). If you use a string name, you may omit the GUID of the property set, and the default value is PS_PUBLIC_STRINGS. If you use the numeric identifier, the GUID of the property set is required.

The correct format of the property set GUID is an undelimited byte string, the same format as required by the CDO [Fields.Item](#) property.

The following example adds a format for a user-defined property, using the string name for the property.

```
' Use GUID of property set (excluding delimiters) and string
const AmPidTag_Location = "{EE0392EFA5B91B10ACC100AA00423326} NAMED_PROP"
Set objFormat = objRenderer.Formats.Add(AmPidTag_Location, Null)
```

The following example adds a format for a user-defined property, using the identifier of the property.

```
' Use GUID of property set (excluding delimiters) and numeric ID
const AmPidTag_Location = "{EE0392EFA5B91B10ACC100AA00423326} 0x7E00"
Set objFormat = objRenderer.Formats.Add(AmPidTag_Location, Null)
```

For information on property sets and GUIDs, see the *MAPI Programmer's Reference*.

Named Formats

Named formats are objects that can be created by the user or defined by the CDO Rendering Library. Like property-only formats, these objects can contain any number of patterns which in turn contain values and RenderUsing pattern strings. They do not require a specific property value.

Once defined, the named format may be used like any other substitution token in a pattern string by placing the name between percent (%) symbols. This makes complex layering possible. You cannot do this with property-only formats.

For instance, setting `RenderingApplication.VirtualRoot` creates the `virtroot` named format, which contains a single "*" pattern containing the IIS virtual-root setting. The string `%virtroot%` in a pattern string will be replaced by the value set in `RenderingApplication.VirtualRoot`.

In this way, named format objects may be used as a table of symbols, containing replacement strings that can be referenced by name. A few reserved named formats contain formatting instructions for specific operations performed by the CDO Rendering Library rendering objects. These reserved names are listed in the following table.

Reserved Named Formats and Usage

| Named format | Used by | Function |
|------------------------|------------------------------------|--|
| virtroot | <u>RenderingApplication</u> object | IIS virtual root containing Web application. |
| classpath | RenderingApplication object | Message class to directory mapping for forms, for instance "Ipm/Note/" for the IPM.Note class. |
| message_Link | <u>ObjectRenderer</u> object | Pattern used by the <u>RenderLink Method (ObjectRenderer Object)</u> . |
| message_Link | <u>ContainerRenderer</u> object | Pattern used as link for objects in a table (messages or address entries). Automatically selects appropriate column based on table cell value. |
| folderhierarchy_Parent | ContainerRenderer object | Pattern used to render the link to the parent of the current folder when rendering a folder hierarchy. |
| folderhierarchy_Icon | ContainerRenderer object | Patterns for icons used when rendering subfolders, and special folders, in the folder hierarchy. |
| folderhierarchy_Link | ContainerRenderer object | Pattern used to render active links to subfolders. |

Important In order to render a hierarchy, the named formats `folderhierarchy_Parent`, `folderhierarchy_Icon`, and `folderhierarchy_Link` must be defined. To know what patterns to add, see the examples in the tables in Suggested Formats and Patterns.

Formats Collection Objects

Three objects contain Formats collection objects: ContainerRenderer, ObjectRenderer, and RenderingApplication. The Formats collection object of the RenderingApplication object is used as a global formats collection. This means that if certain formats will always be used, you can store them in the RenderingApplication object and they will exist for all objects that you may render. This added scope is the reason virtroot and classpath exist in the RenderingApplication object and not in the individual renderers. See Setting virtroot and classpath.

A Formats collection object can contain zero or more format objects.

If a named format in the ObjectRenderer object or the ContainerRenderer object duplicates a named format in the RenderingApplication object, the format in the ObjectRenderer or ContainerRenderer will be used.

About Patterns

A pattern object is an object specifying rendering information for a particular set of values of a property on the object being rendered. Each format contains zero or more pattern objects, and each pattern is associated with a specific MAPI property value being rendered.

Pattern objects control the output of the associated MAPI property values by substituting the designated RenderUsing string if a match is found. The Pattern.Value property accepts any VARIANT type, but it is normally set to a string value. If the MAPI property value being matched contains a nonstring type, the Pattern.Value VARIANT is coerced to that type and a test for equality is performed.

The supported nonstring property types are PT_BOOLEAN, PT_I2, PT_LONG, and PT_BINARY. If the MAPI property is PT_UNICODE, it is matched against the BSTR equivalent of the pattern's value VARIANT using the WILDMAT matching standard, described in the section Pattern Matching. If no match is found, or if no patterns are defined for the format, the property is rendered by type, as illustrated in the following table.

Default Output Styles of Supported Property Types

| Property type | Rendered value |
|---------------|---|
| PT_UNICODE | String is rendered verbatim. |
| PT_MV_UNICODE | Renders all strings separated by a semicolon (;) and a space. |
| PT_SYSTIME | Shortened date format followed by time without seconds (localized by LCID). |
| PT_R4 | Rendered as a stringized float value (localized by LCID). |
| PT_DOUBLE | Rendered as a stringized double value (localized by LCID). |
| PT_BOOLEAN | Renders 1 for True, 0 for False. |
| PT_LONG | Rendered as a stringized long integer value. |
| PT_I2 | |
| PT_ERROR | Ignored if error is MAPI_E_NOT_FOUND, otherwise assumed to be a PT_OBJECT type, which is retrieved with OpenProperty .
Rendered as Unicode string or a hexized binary string, depending on object type. |
| PT_NULL | No output. |

The exceptions to this default property-type handling are listed in the following table.

Rendering Support for Special Properties

| Property | Rendered value |
|--------------------------------|--|
| PR_ENTRYID | Hexized binary string. |
| PR_LONGTERM_ENTRYID_FROM_TABLE | |
| PR_BODY | Invokes RTF2HTML conversion engine. |
| PR_RTF_COMPRESSED | |
| PR_ATTACH_DATA_BIN | Sends MIME data to output stream, or creates a link to public folder object.
Currently only supports ATTACH_BY_VALUE attachment method. |

All other property types are not supported.

Pattern Matching

CDO Rendering library Pattern objects support the WILDMAT syntax of pattern matching. The WILDMAT format was developed to provide a uniform mechanism for matching patterns in the same manner that the UNIX shell matches file names. WILDMAT works as follows:

There are five pattern-matching operations other than a strict one-to-one match between the pattern and the source that must be checked for a match. The first is an asterisk (*) to match any sequence of zero or more characters. The second is a question mark (?) to match any single character.

The third operation specifies a specific set of characters. The set is specified as a list of characters, or as a range of characters where the beginning and end of the range are separated by a minus (or hyphen) character, or as any combination of lists and ranges. The hyphen can also be included in the range as a character if it is at the beginning or end of the range. This set is enclosed in square brackets. The close square bracket (]) may be used in a range if it is the first character in the set.

The fourth operation allows you to exclude characters from the set and is specified by adding a caret character (^) at the beginning of the test string just inside the open square bracket. The final operation uses the backslash (\) character to invalidate the special meaning of an open square bracket ([), the asterisk, or the question mark.

Except for characters in a specified range, all patterns in the CDO Rendering library are matched in a case-insensitive fashion. Example patterns are listed in the following table.

Examples of Valid WILDMAT Patterns

| Pattern | Matches |
|----------------|---|
| a??d | Any four-character string that begins with "a" (or "A") and ends with "d" (or "D"). |
| *bdc | Any string that ends with the string "bdc" (any combination of case, without quotes). |
| [0-9a-zA-Z] | Any alphanumeric character (in English). |
| [^]-] | Any character other than a close square bracket or a minus sign/dash. |

Link Patterns

A link pattern specifies the HTML that wraps text being rendered, which creates a hyperlink. The link pattern could, for example, be in the HTML "HREF" syntax, in order to create a link. Link patterns are most important when rendering an active table of messages or an active hierarchy of folders, to give each message or folder a hot link the Web user can click to obtain more detail, or to read or compose a message.

You should set up link patterns when you create rendering objects, and they will be used throughout the application.

Setting the **LinkPattern** property changes the "message_Link" named format. This format is created with a default value of %value%, which means that initially, it produces no links. Setting **LinkPattern** provides active hyperlinks that specify the location of a form or the links from a message or folder to the next message or folder. The message_Link named format should be changed only by setting the **LinkPattern** property.

For example, the following link pattern specifies the HTML that wraps the contents of a table cell (indicated by %value%) when the objContainerRenderer.Render method is called:

```
objContainerRenderer.LinkPattern = "<A HREF='details.asp?obj=%obj%'>%value%</A>"
```

Another type of link pattern is the "folderhierarchy_Link" format. For more information, see the tables in the section Suggested Formats and Patterns.

RenderUsing Strings

Pattern.RenderUsing strings allow you to specify replacement tokens inside percent symbols (%) to indicate which values should be substituted when the string is constructed. In addition to formatting and rendering the current property value, other information about the MAPI object can be rendered as part of the pattern string.

There are only a few substitution tokens that have special meaning in CDO Rendering. They are listed in the following table.

CDO Rendering Substitution Tokens

| Token | Rendered value |
|-------------|--|
| %kvalue% | For numeric properties, the value/1024. Does not include "KB" symbol. |
| %rowid% | Index of row in the rendered table. Ranges from 0 to RowsPerPage - 1. |
| %obj% | PR_ENTRYID or PR_LONGTERM_ENTRYID_FROM_TABLE of object. |
| %parentobj% | PR_PARENT_ENTRYID of the object. |
| %value% | The value of the property being rendered, based on usage of Format object. |

In the previous table, output is fixed for all but the %value% token. In other words, the output is directly related to the underlying MAPI object set as the **DataSource**. For %value%, however, the rendering output changes based on property type and the semantics of the format object containing the pattern string.

For instance, when %value% appears in the ContainerRenderer.LinkPattern string, it is replaced with the entire contents of the table cell before the link.

Creating Rendering Objects, Formats, and Patterns

The following code shows how to create an ObjectRenderer object for a message. It also adds the following:

- One property-only format object to be used by the column object to render the MAPI property PR_IMPORTANCE. The RenderUsing string contains a reference to a named format that will be used to display language-specific strings.
- The named format object referenced in the RenderUsing string.
- A pattern for each possible value of PR_IMPORTANCE.
- A LinkPattern string to enable the use of the RenderLink Method (ObjectRenderer Object), which renders a link to the DataSource object.

```
' Assume Rendering application is stored in
' Active Server Pages Application object
Set objRenderApp = Application ("RenderingApp")

' Create a renderer object using class object renderer
Set objRenderer = objRenderApp.CreateRenderer (AMHTML_Class_ObjectRenderer)
If (Not objRenderer Is Nothing) Then

    ' Add a property-only Format object (for PR_IMPORTANCE)
    ' (The named format will contain language specific strings.)
    Set objFormat = objRenderer.Formats.Add(CdoPR_IMPORTANCE, Null)
    objFormat.Patterns.Add "*", "%langImportance%"

    ' Add a named Format object to the renderer object
    Set objFormat = objRenderer.Formats.Add(0, "langImportance")

    ' Add patterns to Format object (possible property
    ' values and corresponding RenderUsing strings)
    objFormat.Patterns.Add 0, "Low"
    objFormat.Patterns.Add 1, "Normal"
    objFormat.Patterns.Add 2, "High"

    ' Add LinkPattern string (creates "message_Link" named format)
    objRenderer.LinkPattern = "%virtroot%/forms/%classpath%frmroot.asp?obj=%obj
    %&command=open"

    ' Store new renderer object in Session for later use
    Set Session("ObjectRenderer") = objRenderer
End If
```


Suggested Formats and Patterns

The following tables suggest formats and patterns you can add and use to render a variety of messaging objects. In each table, the first column lists a property-only format or a named format. When the format in the first column contains the value in the second column, it is rendered using the string or image in the third column.

Formats for Rendering Application Object

| Properties on Format Object | Properties on Pattern Object | |
|-----------------------------|------------------------------|---|
| Name or property | Value | RenderUsing |
| "virtroot" | "**" | /exchange |
| "classpath" | "**" | IPM/Note/ and other patterns, as set by FormsRoot |

Formats for ObjectRenderer (DataSource = Message object)

| Properties on Format Object | Properties on Pattern Object | |
|-----------------------------|------------------------------|--|
| Name or property | Value | RenderUsing |
| PR_IMPORTANCE | 0 | "Low" |
| | 1 | "Normal" |
| | 2 | "High" |
| "message_Link" | "**" | %virtroot%/Forms/%classpath%read.asp?obj=%obj% |

In the preceding table, it is assumed that a read.asp file (which displays the message identified by the URL parameter obj) exists in the directory indicated by classpath.

Formats for ContainerRenderer (DataSource = Messages Collection)

| Properties on Format Object | Properties on Pattern Object | |
|-----------------------------|------------------------------|---|
| Name or property | Value | RenderUsing |
| PR_IMPORTANCE | 0 | |
| | 1 | Empty |
| | 2 | |
| PR_MESSAGE_CLASS | "**" | 'envelope.gif'. |
| PR_HASATTACH | 0 | Empty |
| | 1 | |
| PR_MESSAGE_SIZE | "**" | %kvalue%KB |
| "message_Link" | "**" | %virtroot%/Forms/%classpath%read.asp?obj=%obj% |

In the preceding table, it is assumed that a read.asp file (which displays the message identified by the URL parameter obj) exists in the directory indicated by classpath.

Formats for ContainerRenderer (DataSource = Folders Collection)

| Properties on Format Object | Properties on Pattern Object | |
|-----------------------------|------------------------------|--|
|-----------------------------|------------------------------|--|

| Name or property | Value | RenderUsing |
|--------------------------|-------------------------------|---|
| PR_DISPLAY_NAME | InfoStore.RootFolder.
Name | InfoStore.Name |
| "folderhierarchy_Parent" | "" | <TD BGCOLOR=CCCC99
COLSPAN=2> </TD> |
| "folderhierarchy_Icon" | "" | <IMG SRC='folder.gif' ALIGN=CENTER
BORDER=0> |
| "folderhierarchy_Link" | "" |
%value% |

In the preceding table, it is assumed that a folder.asp file (which displays the folder hierarchy using the URL parameter obj as the root folder) exists in the current directory.

Formats for ContainerRenderer (DataSource = AddressEntries Collection)

Properties on Format Object

Properties on Pattern Object

| Name or property | Value | RenderUsing |
|--------------------------|-------------------------------|---|
| PR_DISPLAY_NAME | InfoStore.RootFolder.
Name | InfoStore.Name |
| "folderhierarchy_Parent" | "" | <TD BGCOLOR=CCCC99
COLSPAN=2> </TD> |
| "folderhierarchy_Icon" | "" | <IMG SRC='folder.gif'
ALIGN=CENTER BORDER=0> |
| "folderhierarchy_Link" | "" |
%value% |

In the preceding table, it is assumed that a address.asp file (which displays the messaging user identified by the URL parameter obj) exists in the current directory.

Rendering Objects

This reference contains property and method information for the Microsoft® Collaboration Data Objects (CDO) Rendering Library objects.

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

| Object | Available since version | Properties | Methods |
|---------------------------|-------------------------|--|---|
| <u>CalendarView</u> | 1.2 | BusyCell, BusyIndicator, Categories, Class, Columns, DailyEventCell, DailyTimeCell, FreeBusinessCell, FreeIndicator, FreeNonBusinessCell, Index, Interval, Mode, Name, NumberOfUnits, OOFIndicator, Parent, Source, TentativeIndicator, WeeklyAppointmentCell, WeeklyHeadingCell | IsSameAs, RenderAppointments, RenderDateNavigator, RenderEvents |
| <u>Column</u> | 1.1 | Class, Flags, Index, Name, Parent, Property, RenderUsing, Width | (none) |
| <u>Columns</u> collection | 1.1 | Class, Count, Item, Parent | Add |
| <u>ContainerRenderer</u> | 1.1 | BusinessDayEndTime, BusinessDays, BusinessDayStartTime, CellPattern, Class, CodePage, CurrentStore, CurrentView, DataSource, FirstDayOfWeek, Formats, HeadingCellPattern, HeadingRowPrefix | Render, RenderDate, RenderHeading, RenderProperty, RenderTime |

| | | | |
|-----------------------------|-----|--|---|
| | | HeadingRowSuffix,
Is24HourClock,
LCID, LinkPattern,
Parent,
PrivateStore,
RowsPerPage,
RowPrefix,
RowSuffix,
TablePrefix,
TableSuffix,
TimeZone, Views | |
| <u>Format</u> | 1.1 | Class, Name,
Parent, Patterns,
Property | Delete |
| <u>Formats</u> collection | 1.1 | Class, Count,
Item, Parent | Add |
| <u>ObjectRenderer</u> | 1.1 | Class, CodePage,
DataSource,
Formats, LCID,
LinkPattern,
Parent | RenderDate,
RenderLink,
RenderProperty,
RenderTime |
| <u>Pattern</u> | 1.1 | Class, Parent,
RenderUsing,
Value | Delete |
| <u>Patterns</u> collection | 1.1 | Class, Count,
Item, Parent | Add |
| <u>RenderingApplication</u> | 1.1 | Class, CodePage,
ConfigParameter,
Formats,
FormsRoot, ImplID,
LCID,
LoggingLevel,
Name, Parent,
Version,
VirtualRoot | CreateRenderer,
Impersonate,
ListComposeForms,
LoadConfiguration |
| <u>TableView</u> | 1.1 | Categories, Class,
Columns, Index,
Name, Parent,
Source | IsSameAs |
| <u>Views</u> collection | 1.1 | Class, Count,
Item, Parent | Add |

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.

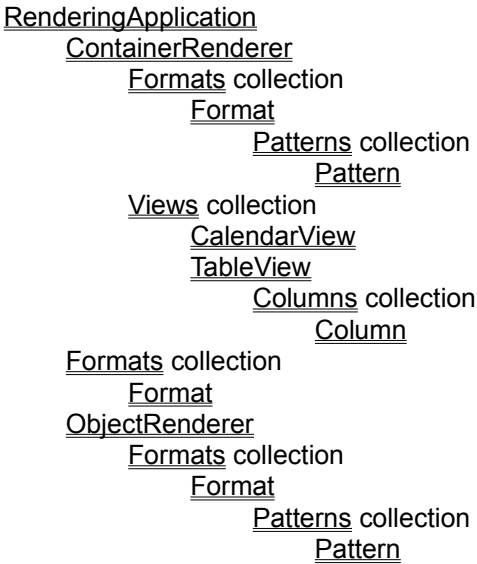
To avoid duplication, the section [Properties Common to All CDO Rendering Library Objects](#) describes the properties that have the same meaning for all CDO Rendering Library objects. These are:

- **Class**

- Parent

Rendering Object Model

The object model for the CDO Rendering Library is hierarchical. The following table shows the containment hierarchy. Each indented object is 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, a Formats collection and a Views collection are both child objects of a ContainerRenderer object, and a RenderingApplication object is a parent object of a ContainerRenderer object. However, a RenderingApplication object is not a parent object of a Views collection.



Properties Common to All CDO Rendering Library Objects

All CDO Rendering Library objects expose the **Class** and **Parent** properties. The **Parent** property indicates the immediate parent of the object, and the **Class** property is an integer value that identifies the CDO Rendering Library object.

Both of these common properties have read-only access in all objects. The RenderingApplication object represents the highest level in the CDO Rendering 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 CDO Rendering Library objects and that have the same meaning for all objects.

Properties

| Name | Type | Access |
|----------------------|--------|-----------|
| <u>Class</u> | Long | Read-only |
| <u>Parent</u> | Object | Read-only |

Class Property (All CDO Rendering Library Objects) 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 CDO Rendering Library object. The following values are defined:

| CDO Rendering Library object | Class value | Type library constant |
|------------------------------|-------------|----------------------------------|
| <u>CalendarView</u> | 12 | CdoClassCalendarView |
| <u>Column</u> | 11 | CdoClassColumn |
| <u>Columns</u> collection | 10 | CdoClassColumns |
| <u>ContainerRenderer</u> | 3 | CdoClassContainerRenderer |
| <u>Format</u> | 4 | CdoClassFormat |
| <u>Formats</u> collection | 5 | CdoClassFormats |
| <u>ObjectRenderer</u> | 2 | CdoClassObjectRenderer |
| <u>Pattern</u> | 7 | CdoClassPattern |
| <u>Patterns</u> collection | 6 | CdoClassPatterns |
| <u>RenderingApplication</u> | 1 | CdoClassApplication |
| <u>TableView</u> | 9 | CdoClassTableView |
| <u>Views</u> collection | 8 | CdoClassViews |

The CDO Rendering Library reserves the value 0 for an object implementing the OLE **IUnknown** interface.

Example

This code fragment traverses a Views collection to find the class of each view object it contains:

```
Dim colViews
Dim objView ' could be any type of view object
If colViews Is Nothing Then
    objResponse.Write "Need to set the Views collection"
End
End If
For Each objView in colViews
    strIndex = str(objView.Index) ' get view's index as a string
    If 9 = objView.Class Then ' CdoClassTableView
        objResponse.Write "View " & strIndex & " is a table view"
    ElseIf 12 = objView.Class Then ' CdoClassCalendarView
        objResponse.Write "View " & strIndex & " is a calendar view"
    Else
```



```
    objResponse.Write "Unknown view type: class " & str(objView.Class)
End If
Next
' error handling here ...
```

Parent Property (All CDO Rendering Library Objects) Group

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 CDO Rendering Library returns the *immediate* parent of an object. The immediate parent for each object is shown in the following table.

| CDO Rendering Library object | Immediate parent in object hierarchy |
|--------------------------------------|--|
| CalendarView | Views collection |
| Column | Columns collection |
| Columns collection | TableView |
| ContainerRenderer | RenderingApplication or Nothing |
| Format | Formats collection |
| Formats collection | ContainerRenderer , ObjectRenderer , or RenderingApplication |
| ObjectRenderer | RenderingApplication or Nothing |
| Pattern | Patterns collection |
| Patterns collection | Format |
| RenderingApplication | Set to Nothing |
| TableView | Views collection |
| Views collection | ContainerRenderer |

The **Parent** property represents the *immediate* parent of the object, rather than the *logical* parent. For example, a [ContainerRenderer](#) object contains a [Views](#) collection, which contains various view objects. The **Parent** property for a view is the immediate parent, the [Views](#) collection, rather than the logical parent, the [ContainerRenderer](#) object.

For more information on the CDO Rendering Library object hierarchy, see [Rendering Object Model](#).

Example

This code fragment displays the **Class** of the **Parent** of a [Formats](#) collection object, which could be any rendering object.

```
Dim colFmts As Formats
Dim parentClass as Long
' assume valid Formats collection
parentClass = colFmts.Parent.Class
MsgBox "Formats parent class = " & parentClass
If 3 = parentClass Then ' CdoClassContainerRender
    ' parent is ContainerRender
Elseif 2 = parentClass Then ' CdoClassObjectRender
```

```
' parent is ObjectRenderer  
Else  
  ' Houston, we have a problem  
End If
```

CalendarView Object

The CalendarView object represents a view of a schedule calendar.

At a Glance

| | |
|----------------------------|------------------------------------|
| Specified in type library: | CDOHTML.DLL |
| First available in: | CDO Rendering Library version 1.2 |
| Parent objects: | Views collection |
| Child objects: | Columns collection |
| Default property: | (none) |

Properties

| Name | Available since version | Type | Access |
|---------------------------------------|-------------------------|--|------------|
| BusyCell | 1.2 | String | Read/write |
| BusyIndicator | 1.2 | String | Read/write |
| Categories | 1.2 | Long | Read/write |
| Class | 1.2 | Long | Read-only |
| Columns | 1.2 | Column object or Columns collection object | Read-only |
| DailyEventCell | 1.2 | String | Read/write |
| DailyTimeCell | 1.2 | String | Read/write |
| FreeBusinessCell | 1.2 | String | Read/write |
| FreeIndicator | 1.2 | String | Read/write |
| FreeNonBusinessCell | 1.2 | String | Read/write |
| Index | 1.2 | Long | Read-only |
| Interval | 1.2 | Long | Read/write |
| Mode | 1.2 | Long | Read/write |
| Name | 1.2 | String | Read-only |
| NumberOfUnits | 1.2 | Long | Read/write |
| OOIndicator | 1.2 | String | Read/write |
| Parent | 1.2 | Views collection object | Read-only |
| Source | 1.2 | Long | Read-only |
| TentativeIndicator | 1.2 | String | Read/write |
| WeeklyAppointmentCell | 1.2 | String | Read/write |
| WeeklyHeadingCell | 1.2 | String | Read/write |

Methods

| Name | Available since version | Parameters |
|----------------------------|-------------------------|---|
| <u>IsSameAs</u> | 1.2 | <i>varView</i> as Object |
| <u>RenderAppointments</u> | 1.2 | (optional) <i>varStartDate</i> as Variant ,
(optional) <i>varResponseObject</i> as Object |
| <u>RenderDateNavigator</u> | 1.2 | (optional) <i>varStartDate</i> as Variant ,
(optional) <i>varMonths</i> as Variant ,
(optional) <i>varResponseObject</i> as Object |
| <u>RenderEvents</u> | 1.2 | (optional) <i>varStartDate</i> as Variant ,
(optional) <i>varResponseObject</i> as Object |

Remarks

A calendar view is a specification of a calendar rendering for a Messages collection in a calendar folder. It is applied to AppointmentItem objects in a Messages collection obtained from a Folder object reserved for calendar data:

```
Dim objSession As Session
Dim objCalendarFolder As Folder
Dim objAppointments As Messages
```

```
Set objCalendarFolder = objSession.GetDefaultFolder_  
                        (CdoDefaultFolderCalendar)
```

```
Set objAppointments = objCalendarFolder.Messages
```

The Messages collection obtained in this manner should contain AppointmentItem objects exclusively. If presented with objects of other classes, the calendar view's rendering methods attempt to render them as if they were appointments, but unexpected results are likely. In any case, objects not exposing the StartTime and EndTime properties are not rendered.

The calendar view is applied to the collection in the context of a ContainerRenderer. The container renderer specifies the Messages collection in its DataSource property and the calendar view to be applied in its CurrentView property.

The calendar view inherits all the functionality of the TableView object, and has additional capability specific to rendering a calendar as a table.

The HTML output of a calendar view is constructed on a principal unit of display, or *time unit*, which is specified by the view's Mode property. The time unit can be a day or a week. The view's *time span*, determined by its NumberOfUnits property, represents the total amount of calendar time displayed in a single HTML page. Each column of the view covers exactly one day, no matter what the overall time unit and time span have been set to. A column is subdivided into individual cells or *time slots*. The size of these subdivisions is controlled by the view's Interval property.

A calendar view is normally generated externally to a CDO application, although a nonpersistent calendar view can be created with the Add method of the Views collection. A calendar view created in this way ceases to exist when the collection is released.

BusyCell Property (CalendarView Object) Group

The **BusyCell** property returns or sets a pattern string for rendering an appointment. Read/write.

Syntax

objCalendarView.**BusyCell**

Data Type

String

Remarks

The **BusyCell** property is used to specify rendering information for [AppointmentItem](#) objects. The **BusyCell** property applies only when the **Mode** property of the calendar view contains **CdoModeCalendarDaily**.

The default value of the **BusyCell** property is

```
<td rowspan=%apptlength% colspan=%apptwidth% bgcolor=ffffff valign=top align=left width=%percentwidth%>%value%</td>
```

for a daily view. For a weekly view, use the [WeeklyAppointmentCell](#) property.

For more information on substitution tokens, see the [RenderUsing](#) property of the [Pattern](#) object.

BusyIndicator Property (CalendarView Object) Group

The **BusyIndicator** property returns or sets a pattern string for rendering the indicator bar for a busy time period. Read/write.

Syntax

objCalendarView.**BusyIndicator**

Data Type

String

Remarks

The **BusyIndicator** property contains the rendering for the indicator bar for a busy time slot. The indicator bar is the thin vertical column between the time column and the appointments table.

BusyIndicator applies only when the **Mode** property of the calendar view contains **CdoModeCalendarDaily**.

The default value of the **BusyIndicator** property is

```
<td rowspan=%apptlength% colspan=1 bgcolor=0000ff width=%percentwidth%>&nbsp;</td>
```

For more information on substitution tokens, see the **RenderUsing** property of the [Pattern](#) object.

Categories Property (CalendarView Object)

The **Categories** property returns or sets the number of categories in this calendar view. Read/write.

Syntax

objCalendarView.Categories

Data Type

Long

Remarks

The current version of the CDO Rendering Library does not support a tabular view of a calendar. Therefore, the **Categories** property is currently ignored on a CalendarView object.

Columns Property (CalendarView Object) Group

The **Columns** property returns a single Column object or a Columns collection for this calendar view. Read-only.

Syntax

Set *objColumns* = *objCalendarView*.**Columns**

Set *objColumn* = *objCalendarView*.**Columns**(*index*)

objColumns

Object. The Columns collection of this calendar view.

objCalendarView

Required. The CalendarView object.

objColumn

Object. An individual Column object belonging to this calendar view's Columns collection.

index

Integer. An index into the calendar view's Columns collection.

Data Type

Object (Column or Columns collection)

Remarks

If a Column object is to be accessed with the *index* parameter, the value of *index* must be between 1 and the size of the CalendarView object's Columns collection. This size is available in the collection's Count property.

Although the **Columns** property itself is read-only, the collection it returns can be accessed in the normal manner through its **Add** method, and the properties on its member Column objects retain their respective read/write or read-only accessibility.

A calendar view renders each day as a one-column-wide HTML table. For each time slot, the columns specified in the Columns collection are rendered as successive strings within a single HTML table cell. When you add a column to the collection, you are not causing an additional HTML table column to be rendered. Instead, you are causing the column's contents to be concatenated with the strings for all the other columns in the table cell.

DailyEventCell Property (CalendarView Object) Group

The **DailyEventCell** property returns or sets a pattern string for rendering an event. Read/write.

Syntax

objCalendarView.**DailyEventCell**

Data Type

String

Remarks

The **DailyEventCell** property is used to specify rendering information for individual events when the **RenderEvents** method is called. The **DailyEventCell** property applies only when the **Mode** property of the calendar view contains **CdoModeCalendarDaily**.

The default value of the **DailyEventCell** property is

```
<td rowspan=%apptlength% colspan=%apptwidth% bgcolor=c0c0c0 valign=top align=left width=%percentwidth%>%value%</td>
```

For more information on substitution tokens, see the **RenderUsing** property of the **Pattern** object.

DailyTimeCell Property (CalendarView Object) Group

The **DailyTimeCell** property returns or sets a pattern string for rendering the time column. Read/write.

Syntax

objCalendarView.**DailyTimeCell**

Data Type

String

Remarks

The **DailyTimeCell** property is used to specify rendering information for the time slot cells in the column at the left of a daily view. The **DailyTimeCell** property applies only when the **Mode** property of the calendar view contains **CdoModeCalendarDaily**.

The default value of the **DailyTimeCell** property is

```
<td bgcolor=c0c0c0 valign=middle align=right width=%percentwidth%><font size=-1><nobr>%time%</nobr></font></td>
```

For more information on substitution tokens, see the **RenderUsing** property of the Pattern object.

FreeBusinessCell Property (CalendarView Object) Group

The **FreeBusinessCell** property returns or sets a pattern string for rendering a free time slot during business hours. Read/write.

Syntax

objCalendarView.FreeBusinessCell

Data Type

String

Remarks

The **FreeBusinessCell** property applies only when the **Mode** property of the calendar view contains **CdoModeCalendarDaily**.

The default value of the **FreeBusinessCell** property is

```
<td rowspan=%apptlength% colspan=%apptwidth% bgcolor=ffffff valign=top align=left width=%percentwidth%><br></td>
```

for a daily view. For a weekly view, use the **WeeklyHeadingCell** property.

For more information on substitution tokens, see the **RenderUsing** property of the **Pattern** object.

FreeIndicator Property (CalendarView Object) Group

The **FreeIndicator** property returns or sets a pattern string for rendering the indicator bar for a free time period. Read/write.

Syntax

objCalendarView.FreeIndicator

Data Type

String

Remarks

The **FreeIndicator** property contains the rendering for the indicator bar for a free time slot. The indicator bar is the thin vertical column between the time column and the appointments table.

FreeIndicator applies only when the **Mode** property of the calendar view contains **CdoModeCalendarDaily**.

The default value of the **FreeIndicator** property is

```
<td rowspan=%apptlength% colspan=1 bgcolor=ffffff width=%percentwidth%>&nbsp;</td>
```

For more information on substitution tokens, see the **RenderUsing** property of the Pattern object.

FreeNonBusinessCell Property (CalendarView Object) Group

The **FreeNonBusinessCell** property returns or sets a pattern string for rendering a free time slot outside of business hours. Read/write.

Syntax

objCalendarView.FreeNonBusinessCell

Data Type

String

Remarks

The default value of the **FreeNonBusinessCell** property is

```
<td rowspan=%apptlength% colspan=%apptwidth% bgcolor=c0c0c0 valign=top align=left width=%percentwidth%><br></td>
```

for a daily view and

```
<tr>
```

for a weekly view. For more information on substitution tokens, see the **RenderUsing** property of the **Pattern** object.

Index Property (CalendarView Object)

Group

The **Index** property returns the index number for this CalendarView object within the Views collection. Read-only.

Syntax

objCalendarView.**Index**

Data Type

Long

Remarks

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

The first view in the Views collection has a **Index** value of 1.

An index value should not be considered a static value that remains constant for the duration of a container renderer. It can be affected when other views are added and deleted.

Interval Property (CalendarView Object)

Group

The **Interval** property returns or sets the length of a time slot in minutes. Read/write.

Syntax

objCalendarView.Interval

Data Type

Long

Remarks

The **Interval** property is forced by the CDO Rendering Library to a value between 5 and 60 that divides integrally into 60. The possible resulting values are 5, 6, 10, 12, 15, 20, 30, and 60. The default value of **Interval** is 30 minutes.

The **Interval** property applies only when the **Mode** property of the calendar view contains **CdoModeCalendarDaily**. If **Mode** is **CdoModeCalendarWeekly**, time slots are not displayed, and each AppointmentItem object is rendered with its **StartTime** and **EndTime** values.

IsSameAs Method (CalendarView Object) Group

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

Syntax

```
boolSame = objCalendarView.IsSameAs(varView)
```

boolSame

On successful return, contains **True** if the two objects are the same.

objCalendarView

Required. This CalendarView object.

varView

Required. Object. The view object being compared against.

Remarks

The *varView* parameter should be declared as an **Object** rather than as a **CalendarView**. This allows for comparison among different classes of view objects being held in a Views collection.

Two view objects are considered to be the same if and only if their pointer values are the same, that is, if and only if they are the identical object. Otherwise **IsSameAs** returns **False**.

Mode Property (CalendarView Object)



The **Mode** property returns or sets the time unit of the calendar view. Read/write.

Syntax

objCalendarView.**Mode**

Data Type

Long

Remarks

The **Mode** property specifies the principal unit of display on which the calendar rendering is based. It controls the style of views generated by the [RenderAppointments](#) and [RenderEvents](#) methods. Together with the [NumberOfUnits](#) property it determines the overall time span of the calendar view, that is, the total amount of time rendered onto one HTML page.

Mode can have exactly one of the following values:

| Mode setting | Decimal value | Time unit |
|------------------------------|----------------------|--|
| CdoModeCalendarDaily | 0 | This calendar view is rendered in multiples of a day. |
| CdoModeCalendarWeekly | 1 | This calendar view is rendered in multiples of a week. |

The **Mode** property defaults to **CdoModeCalendarDaily**.

Name Property (CalendarView Object)

Group

The **Name** property returns the display name of this CalendarView object. Read-only.

Syntax

objCalendarView.Name

Data Type

String

Remarks

The **Name** property represents the display name assigned to this calendar view. It can be used to refer to the calendar view, and to retrieve it by name using the container renderer's **CurrentView** property or the **Views** collection's **Item** property.

The names of the predefined calendar views are "Daily" for a view with mode **CdoModeCalendarDaily** and "Weekly" for a view with mode **CdoModeCalendarWeekly**. The mode of the view is available from its **Mode** property.

You should give every view object a unique name. CDO does not enforce uniqueness, but if a collection has duplicate names, only the first one can be found by name.

NumberOfUnits Property (CalendarView Object) Group

The **NumberOfUnits** property returns or sets the number of time units to include in a rendering. Read/write.

Syntax

objCalendarView.NumberOfUnits

Data Type

Long

Remarks

The **NumberOfUnits** and **Mode** properties together determine the overall time span of the calendar view, that is, the total amount of time rendered onto one HTML page. **NumberOfUnits** defaults to 1.

For example, if **Mode** contains **CdoModeCalendarWeekly** and **NumberOfUnits** is set to 2, each output page displays two weeks of calendar time.

OOFIndicator Property (CalendarView Object) Group

The **OOFIndicator** property returns or sets a pattern string for rendering the indicator bar for an out-of-office (OOF) time period. Read/write.

Syntax

objCalendarView.OOFIndicator

Data Type

String

Remarks

The **OOFIndicator** property contains the rendering for the indicator bar for an out-of-office time slot. The indicator bar is the thin vertical column between the time column and the appointments table.

OOFIndicator applies only when the **Mode** property of the calendar view contains **CdoModeCalendarDaily**.

The default value of the **OOFIndicator** property is

```
<td rowspan=%apptlength% colspan=1 bgcolor=660066 width=%percentwidth
%>&nbsp;</td>
```

For more information on substitution tokens, see the **RenderUsing** property of the Pattern object.

RenderAppointments Method (CalendarView Object) Group

The **RenderAppointments** method renders [AppointmentItem](#) objects in the [Messages](#) collection.

Syntax

```
strHTML = objCalendarView.RenderAppointments( [varStartDate] [, varResponseObject] )
```

strHTML

On successful return, contains a string with the HTML hypertext representing the appointments. However, if the *varResponseObject* parameter is supplied, **RenderAppointments** returns a value of **Empty**.

objCalendarView

Required. This CalendarView object.

varStartDate

Optional. Variant (**vbDate** format). The date/time used to determine the starting date from which to render appointments. The time portion of *varStartDate* is ignored. The default value is the current date.

varResponseObject

Optional. Object. An Active Server response object used to accumulate HTML output to send to the browser. This parameter is used primarily in ASP applications. If *varResponseObject* is not supplied, the output is written to *strHTML*.

Remarks

The **RenderAppointments** method renders appointments starting with a date calculated from the *varStartDate* parameter, but not necessarily equal to its value. The rendering starts at the beginning of the time unit containing the *varStartDate* value. For example, if the calendar view's **Mode** property is set to **CdoModeCalendarWeekly**, the [ContainerRenderer](#) object's **FirstDayOfWeek** property is set to **CdoMonday**, and the value of *varStartDate* indicates a Sunday, appointments are rendered starting with the preceding Monday.

The **RenderAppointments** method generates an HTML table containing one or more cells. If the **Mode** property is **CdoModeCalendarDaily**, the table contains time slots showing free and busy times for the day. The length of the time slots is determined by the **Interval** property. All-day events are shown by the [RenderEvents](#) method and do not appear in the **RenderAppointments** table.

If **Mode** is **CdoModeCalendarWeekly**, the table contains a list of appointments and events for each of seven days, starting with the day designated in the container renderer's **FirstDayOfWeek** property.

Appointments are rendered for the number of time units indicated by the **NumberOfUnits** property, beginning with the starting date calculated from the *varStartDate* parameter. If **NumberOfUnits** is greater than 1, each appointments table appears as a cell within an outer table. If the **Mode** property is **CdoModeCalendarDaily**, the outer table is a single row with **NumberOfUnits** columns. If **Mode** is **CdoModeCalendarWeekly**, the outer table is a single column with **NumberOfUnits** rows.

RenderDateNavigator Method (CalendarView Object) Group

The **RenderDateNavigator** method renders a date navigator that can be used to select a starting date for rendering.

Syntax

```
strHTML = objCalendarView.RenderDateNavigator( [varStartDate] [, varMonths] [, varResponseObject] )
```

strHTML

On successful return, contains a string with the HTML hypertext representing the date navigator. However, if the *varResponseObject* parameter is supplied, **RenderDateNavigator** returns a value of **Empty**.

objCalendarView

Required. This CalendarView object.

varStartDate

Optional. Variant (**vbDate** format). The date/time used to determine the starting date from which to render the date navigator. The time portion of *varStartDate* is ignored. The default value is the current date.

varMonths

Optional. Long. The number of months for which to render the date navigator. The default value is 2.

varResponseObject

Optional. Object. An Active Server response object used to accumulate HTML output to send to the browser. This parameter is used primarily in ASP applications. If *varResponseObject* is not supplied, the output is written to *strHTML*.

Remarks

The **RenderDateNavigator** method renders the date navigator starting with a date calculated from the *varStartDate* parameter, but not necessarily equal to its value. The rendering starts at the beginning of the month containing the *varStartDate* value.

The **RenderDateNavigator** method generates an HTML table containing one or more cells. It renders a date navigator that can be used to choose the date to view with the **RenderAppointments** or **RenderEvents** method. An entire month is displayed at a time.

Every day of a month is rendered individually within the image of the month. Each day's rendering contains a URL calling a JavaScript function with a paradigm of **gotoDate(year, month, day)**. The frame containing the date navigator must define and implement the **gotoDate** function for the linking to work correctly. Microsoft® JScript™ can be used to implement **gotoDate**.

RenderDateNavigator also requires the two image files LEFT.GIF and RIGHT.GIF for the arrow pointers used to move to the previous or next month. You must furnish these in the same directory that contains the .ASP file calling **RenderDateNavigator**. You can define your own .GIF files or use the ones provided with Microsoft® Outlook™ Web Access (OWA), typically in a directory with a path similar to \\...\Exchsrvr\webdata*<lang>*\calendar, where *<lang>* is a language node such as usa, frn, or jpn.

The date navigator is rendered for the number of months specified in the *varMonths* parameter, beginning with the month calculated from the *varStartDate* parameter. If *varMonths* is greater than 1, each month appears as a cell within an outer table. The outer table is a single column with *varMonths* rows.

RenderEvents Method (CalendarView Object) Group

The **RenderEvents** method renders the events in the [Messages](#) collection.

Syntax

```
strHTML = objCalendarView.RenderEvents( [varStartDate] [, varResponseObject] )
```

strHTML

On successful return, contains a string with the HTML hypertext representing the events. However, if the *varResponseObject* parameter is supplied, **RenderEvents** returns a value of **Empty**.

objCalendarView

Required. This CalendarView object.

varStartDate

Optional. Variant (**vbDate** format). The date/time used to determine the starting date from which to render events. The time portion of *varStartDate* is ignored. The default value is the current date.

varResponseObject

Optional. Object. An Active Server response object used to accumulate HTML output to send to the browser. This parameter is used primarily in ASP applications. If *varResponseObject* is not supplied, the output is written to *strHTML*.

Remarks

The **RenderEvents** method renders events starting with a date calculated from the *varStartDate* parameter, but not necessarily equal to its value. The rendering starts at the beginning of the time unit containing the *varStartDate* value. For example, if the calendar view's **Mode** property is set to **CdoModeCalendarWeekly**, the [ContainerRenderer](#) object's **FirstDayOfWeek** property is set to **CdoSunday**, and the value of *varStartDate* indicates a Friday, events are rendered starting with the preceding Sunday.

The **RenderEvents** method generates an HTML table containing one or more cells. If the **Mode** property is **CdoModeCalendarDaily**, the table contains the date being rendered in the first row and events for that day, if any, in subsequent rows.

If **Mode** is **CdoModeCalendarWeekly**, the table contains a single row with the starting and ending dates for the week being rendered. The starting date is calculated using the **FirstDayOfWeek** property. No events are rendered because there are no weekly events.

Events are rendered for the number of time units indicated by the **NumberOfUnits** property, beginning with the starting date calculated from the *varStartDate* parameter. If **NumberOfUnits** is greater than 1, each events table appears as a cell within an outer table. The outer table is a single row with **NumberOfUnits** columns.

Source Property (CalendarView Object)

Group

The **Source** property returns the type of this calendar view. Read-only.

Syntax

objCalendarView.Source

Data Type

Long

Remarks

The **Source** property indicates the source of the definition of the calendar view. It can have exactly one of the following values:

| Calendar view source | Decimal value | Meaning |
|----------------------|---------------|--|
| CdoViewCommon | 0 | This calendar view is predefined globally for all folders and all messaging users. |
| CdoViewCustom | 2 | This calendar view has been defined in the context of the current setting of the DataSource property of the <u>ContainerRenderer</u> object. However, it is still available when the DataSource property is changed. |

For more information on calendar view rendering, see [Rendering Container Objects](#).

TentativeIndicator Property (CalendarView Object) Group

The **TentativeIndicator** property returns or sets a pattern string for rendering the indicator bar for a tentatively busy time period. Read/write.

Syntax

objCalendarView.**TentativeIndicator**

Data Type

String

Remarks

The **TentativeIndicator** property contains the rendering for the indicator bar for a tentatively busy time slot. The indicator bar is the thin vertical column between the time column and the appointments table.

TentativeIndicator applies only when the **Mode** property of the calendar view contains **CdoModeCalendarDaily**.

The default value of the **TentativeIndicator** property is

```
<td rowspan=%apptlength% colspan=1 bgcolor=99ccff width=%percentwidth%>&nbsp;</td>
```

For more information on substitution tokens, see the **RenderUsing** property of the **Pattern** object.

WeeklyAppointmentCell Property (CalendarView Object) Group

The **WeeklyAppointmentCell** property returns or sets a pattern string for rendering an appointment in a weekly view. Read/write.

Syntax

objCalendarView.**WeeklyAppointmentCell**

Data Type

String

Remarks

The **WeeklyAppointmentCell** property is used to specify rendering information for [AppointmentItem](#) objects. The **WeeklyAppointmentCell** property applies only when the **Mode** property of the calendar view contains **CdoModeCalendarWeekly**.

The default value of the **WeeklyAppointmentCell** property is

```
<td rowspan=%daylength% bgcolor=ffffff valign=top align=left height=24><font size=2>%value%</font></td>
```

For more information on substitution tokens, see the [RenderUsing](#) property of the [Pattern](#) object.

WeeklyHeadingCell Property (CalendarView Object) Group

The **WeeklyHeadingCell** property returns or sets a pattern string for rendering a day heading in a weekly view. Read/write.

Syntax

objCalendarView.**WeeklyHeadingCell**

Data Type

String

Remarks

The **WeeklyHeadingCell** property is used to specify rendering information for the heading cells of a weekly view, which indicate the days of the week. The **WeeklyHeadingCell** property applies only when the **Mode** property of the calendar view contains **CdoModeCalendarWeekly**.

The default value of the **WeeklyHeadingCell** property is

```
<td align=right bgcolor=c0c0c0 height=8 width=50%><b><font size=2>%daylabel  
%</font></b></td>
```

For more information on substitution tokens, see the **RenderUsing** property of the Pattern object.

Column Object

The Column object represents a column within a view.

At a Glance

| | |
|----------------------------|-----------------------------------|
| Specified in type library: | CDOHTML.DLL |
| First available in: | CDO Rendering Library version 1.1 |
| Parent objects: | <u>Columns</u> collection |
| Child objects: | (none) |
| Default property: | (none) |

Properties

| Name | Available since version | Type | Access |
|--------------------|-------------------------|---------------------------|------------|
| <u>Class</u> | 1.1 | Long | Read-only |
| <u>Flags</u> | 1.1 | Long | Read/write |
| <u>Index</u> | 1.1 | Long | Read-only |
| <u>Name</u> | 1.1 | String | Read/write |
| <u>Parent</u> | 1.1 | Columns collection object | Read-only |
| <u>Property</u> | 1.1 | Long or String | Read-only |
| <u>RenderUsing</u> | 1.1 | String | Read/write |
| <u>Width</u> | 1.1 | Long | Read/write |

Methods

(None.)

Flags Property (Column Object)

Group

The **Flags** property returns or sets the flags specifying certain display attributes of this Column object. Read/write.

Syntax

objColumn.Flags

Data Type

Long

Remarks

The **Flags** property contains the following bits, which can be set in any combination:

| Attribute flag | Decimal value | Meaning |
|-----------------------------|---------------|---|
| CdoColumnBitmap | 8 | The property rendered in this column is displayed using a bitmap. |
| CdoColumnNotSortable | 32 | The display cannot be sorted on the property rendered in this column. |

If the **CdoColumnBitmap** flag is set, the column width in the **Width** property is expressed in pixels instead of characters. If you use a bitmap, be sure you set **Width** large enough to render the .GIF image, or you may get unexpected results.

Columns predefined in common, folder, and personal views have 12106 = &H2F4A in the **Flags** property. This default setting includes the **CdoColumnBitmap** flag together with several obsolete flags that are ignored by the current version of CDO. If you change this value by setting **Flags**, it is immaterial whether or not you preserve the obsolete flags from the default value

If you are rendering a calendar view, the value of the **Flags** property is ignored. The [CalendarView](#) object makes its own alignment calculations.

Index Property (Column Object)

Group

The **Index** property returns the index number for this Column object within the Columns collection. Read-only.

Syntax

objColumn.Index

Data Type

Long

Remarks

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

The first column in the collection has an **Index** value of 1.

An index value should not be considered a static value that remains constant for the duration of a view. It can be affected when other columns are added and deleted.

Name Property (Column Object)

Group

The **Name** property returns or sets the default heading for this Column object. Read/write.

Syntax

objColumn.Name

Data Type

String

Remarks

The **Name** property represents the text to be rendered for this column in the header row. The column header is rendered according to the **HeadingCellPattern** property of the ContainerRenderer object.

The **Name** property is also useful for distinguishing between columns that render the same property. Two different columns, for example, could render the **CdoPR_MESSAGE_CLASS** property, one as an icon and the other as a text string. The two columns would have the same **Property** property, but they should have different **Name** properties, such as "Icon" and "Message Class".

Property Property (Column Object)

Group

The **Property** property returns the name or tag of the property rendered in this column. Read-only.

Syntax

objColumn.Property

Data Type

Variant (Long or String)

Remarks

The **Property** property is a long integer if the column property was specified by property tag. If it was specified by name, the **Property** property is a string. Predefined MAPI properties are always specified by property tag. Custom named properties can be specified either by tag or by name. The manner of specification depends on the setting of the *varProperty* parameter of the Columns collection's **Add** method. For example, if the column property being added is represented by a Field object, *varProperty* can be set to the field's **ID** property to specify by tag, or to the field's **Name** property to specify by name.

If the column property was specified by name, the property name in the string can optionally be prefixed with a GUID string identifying its property set. In this case, the GUID should be enclosed in braces. For more information and examples, see Using Named Properties.

RenderUsing Property (Column Object)

Group

The **RenderUsing** property returns or sets a rendering source that determines how a column property is rendered. Read/write.

Syntax

objColumn.RenderUsing

Data Type

Variant (String)

Remarks

The **RenderUsing** property provides a source for rendering the column property into HTML hypertext. The column property is designated in the column's **Property** property.

If no column rendering information is available, the container renderer searches for a Format object representing the renderable property. If no such format can be found, or if the format contains no Pattern object appropriate for the renderable property's value, the property is rendered by default according to its data type and value.

If the **RenderUsing** string contains substitution tokens within percent signs, such as %value%, the tokens are replaced by the appropriate attributes of the column property to generate the HTML hypertext. If there are no substitution tokens in the string, the string itself is rendered without modification.

For more information on rendering sources and substitution tokens, see the Pattern object's **RenderUsing** property.

Width Property (Column Object)

Group

The **Width** property returns or sets the width for this Column object. Read/write.

Syntax

objColumn.Width

Data Type

Long

Remarks

The **Width** property represents the horizontal space the column is to occupy when displayed from the HTML output of the rendering.

For common, folder, and personal views, the column widths are relative. They are specified in integer units typically representing characters. The final display width is arrived at in the following manner:

1. The container renderer determines the overall width of the table view by adding together the **Width** properties of every column in the Columns collection of the TableView object.
2. The container renderer computes the proportional width for each column by dividing its **Width** property by the overall table view width.
3. The proportional width for each column is placed in the HTML output.
4. The browser calculates each column's final display width from its proportional width and the available horizontal space in the browser window.

For custom views, the column widths are absolute and are specified in characters. However, if the **CdoColumnBitmap** flag is set in a column's Flags property, that column's width is expressed in pixels.

If you are rendering a calendar view, the value of the **Width** property is ignored. The CalendarView object makes its own calculations for column widths based on the values of its Mode and NumberOfUnits properties.

Columns Collection Object

The Columns collection object contains zero or more columns in a view.

At a Glance

Specified in type library: CDOHTML.DLL
First available in: CDO Rendering Library version 1.1
Parent objects: [CalendarView](#)
[TableView](#)
Child objects: [Column](#)
Default property: [Item](#)

A Columns collection supports count and index values that let you access an individual Column object through the **Item** property. The Columns collection also supports the Microsoft® Visual Basic® **For Each** statement.

Properties

| Name | Available since version | Type | Access |
|------------------------|-------------------------|------------------|-----------|
| Class | 1.1 | Long | Read-only |
| Count | 1.1 | Long | Read-only |
| Item | 1.1 | Column object | Read-only |
| Parent | 1.1 | TableView object | Read-only |

Methods

| Name | Available since version | Parameters |
|---------------------|-------------------------|--|
| Add | 1.1 | <i>pszDisplayName</i> as String ,
<i>varProperty</i> as Variant ,
<i>IWidth</i> as Long ,
<i>IFlags</i> as Long ,
<i>IInsertAfter</i> as Long ,
(optional) <i>varType</i> as Long |

Add Method (Columns Collection)

Group

The **Add** method creates and returns a new Column object in the Columns collection.

Syntax

Set *objColumn* = *objColumnsColl*.**Add**(*pszDisplayName*, *varProperty*, *IWidth*, *IFlags*, *IInsertAfter* [, *varType*])

objColumn

On successful return, contains the new Column object.

objColumnsColl

Required. The Columns collection object.

pszDisplayName

Required. String. The display name to be assigned to the new Column object.

varProperty

Required. Variant (Long or String). The property tag for the predefined or user-defined property, or the custom name of the user-defined property, that is to be rendered in the new Column object.

IWidth

Required. Long. The absolute width of the new Column object. An added column is always handled as if it were in a custom view, even if it is added to a persistent view, so the width is never treated as relative.

IFlags

Required. Long. The flags specifying certain display attributes of the new Column object. For a description of these flags, see the Column object's **Flags** property.

IInsertAfter

Required. Long. The **Index** value of the column after which the new Column object is to be added to the collection. If this parameter is 0, the new column is added at the beginning of the collection. If *IInsertAfter* is greater than the size of the collection, the new column is added at the end.

varType

Optional (required for user-defined properties). Long. The property type of the property to be rendered in the new Column object. The *varType* parameter is only needed for custom properties, because the property type of a predefined property is already included in its property tag, which is passed in the *varProperty* parameter.

Remarks

The *pszDisplayName*, *varProperty*, *IWidth*, *IFlags*, and *IInsertAfter* parameters correspond to the **Name**, **Property**, **Width**, **Flags**, and **Index** properties of the new Column object, except that the new column's **Index** is greater by 1 than the *IInsertAfter* parameter.

The first column in the collection has an **Index** value of 1, and the last has an **Index** equal to the size of the collection in the **Count** property. If the value of the *IInsertAfter* parameter exceeds the collection's size, the new column is added with an **Index** greater by 1 than the previous size. If *IInsertAfter* is less than the previous size, the **Index** values of all columns after the new column are incremented by 1. In all cases, the collection's **Count** is incremented by 1.

The *varProperty* parameter designates the property to be rendered in the column. The parameter can be a long integer specifying the property by property tag, or a string specifying it by custom name. In both cases it corresponds to the **Property** property. Predefined properties are always specified by tag, while user-defined properties can be specified by either tag or name.

If the *varProperty* parameter is a custom property name, it can optionally be prefixed with a GUID string identifying its property set. In this case, the GUID should be enclosed in braces. For more information and examples, see Using Named Properties.

The allowable values for the *varType* parameter are listed in the Field object's **Type** property.

If you are rendering a calendar view, the value of the *Width* parameter is ignored, although it is still required. The CalendarView object makes its own calculations for column widths based on the values of its **Mode** and **NumberOfUnits** properties.

Count Property (Columns Collection)

Group

The **Count** property returns the number of [Column](#) objects in the collection. Read-only.

Syntax

objColumnsColl.Count

Data Type

Long

Remarks

For more information on using the **Count** and **Item** properties, see the example in the [Item](#) property.

Item Property (Columns Collection)

Group

The **Item** property returns the specified Column object from the Columns collection. Read-only.

Syntax

objColumnsColl.**Item**(*index*)

objColumnsColl.**Item**(*propTag*)

index

Integer. Value must be greater than 0 and less than or equal to 65,535 (&HFFFF). Specifies the index within the Columns collection.

propTag

Long. Value must be greater than or equal to 65,536 (&H10000). Specifies the 32-bit property tag of the renderable property corresponding to a column in the collection. The renderable property is indicated in the Column object's Property property.

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

Data Type

Column object

Remarks

The **Item** property works like an accessor property.

If the specified Column object is not found in the collection, the **Item** property returns **Nothing**.

Although the **Item** property itself is read-only, the Column object it returns can be accessed in the normal manner, and its properties retain their respective read/write or read-only accessibility.

Example

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

```
' Put all column names in a collection into a string array
Dim strItemName(100) as String
Dim i As Integer ' loop counter
' error handling omitted from this fragment ...
For i = 1 To objColumnsColl.Count
    strItemName(i) = objColumnsColl.Item(i).Name
    ' or = objColumnsColl(i) since Item and Name are default properties
    If 100 = i Then ' max size of string array
        Exit Function
    End If
Next i
```


ContainerRenderer Object

The ContainerRenderer object renders the rows of a container object as an HTML table.

At a Glance

| | |
|----------------------------|--|
| Specified in type library: | CDOHTML.DLL |
| First available in: | CDO Rendering Library version 1.1 |
| Parent objects: | RenderingApplication (or none) |
| Child objects: | Formats collection
Views collection |
| Default property: | (none) |

Properties

| Name | Available since version | Type | Access |
|---|-------------------------|--|------------|
| <u>BusinessDayEndTime</u> | 1.2 | Variant (vbDate format) | Read/write |
| <u>BusinessDays</u> | 1.2 | Long | Read/write |
| <u>BusinessDayStartTime</u> | 1.2 | Variant (vbDate format) | Read/write |
| <u>CellPattern</u> | 1.1 | String | Read/write |
| <u>Class</u> | 1.1 | Long | Read-only |
| <u>CodePage</u> | 1.1 | Long, Object, or String | Read/write |
| <u>CurrentStore</u> | 1.2 | InfoStore object | Write-only |
| <u>CurrentView</u> | 1.1 | TableView object | Read/write |
| <u>DataSource</u> | 1.1 | AddressEntries collection object, Folders collection object, Messages collection object, or Recipients collection object | Read/write |
| <u>FirstDayOfWeek</u> | 1.2 | Long | Read/write |
| <u>Formats</u> | 1.1 | Format object or Formats collection object | Read-only |
| <u>HeadingCellPattern</u> | 1.1 | String | Read/write |
| <u>HeadingRowPrefix</u> | 1.1 | String | Read/write |
| <u>HeadingRowSuffix</u> | 1.1 | String | Read/write |
| <u>Is24HourClock</u> | 1.2 | Boolean | Read/write |
| <u>LCID</u> | 1.1 | Long | Read-only |
| <u>LinkPattern</u> | 1.1 | String | Read/write |
| <u>Parent</u> | 1.1 | Object; set to Nothing | Read-only |
| <u>PrivateStore</u> | 1.1 | InfoStore object | Write-only |

| | | | |
|--------------------|-----|---|------------|
| <u>RowsPerPage</u> | 1.1 | Integer | Read/write |
| <u>RowPrefix</u> | 1.1 | String | Read/write |
| <u>RowSuffix</u> | 1.1 | String | Read/write |
|
 | | | |
| <u>TablePrefix</u> | 1.1 | String | Read/write |
| <u>TableSuffix</u> | 1.1 | String | Read/write |
| <u>TimeZone</u> | 1.2 | Long | Read/write |
| <u>Views</u> | 1.1 | CalendarView object,
TableView object, or
Views collection object | Read-only |

Methods

| Name | Available since version | Parameters |
|-----------------------|-------------------------|--|
| <u>Render</u> | 1.1 | <i>iStyle</i> as Long ,
(optional) <i>varPageNo</i> as Long ,
(optional) <i>varboolRaw</i> as Boolean ,
(optional) <i>varResponseObject</i> as Object |
| <u>RenderDate</u> | 1.2 | <i>varDate</i> as Variant ,
<i>varFormat</i> as String ,
(optional) <i>varResponseObject</i> as Object |
| <u>RenderHeading</u> | 1.1 | (optional) <i>bstrCellPattern</i> as String ,
(optional) <i>varResponseObject</i> as Object |
| <u>RenderProperty</u> | 1.1 | <i>varProperty</i> as Variant ,
(optional) <i>varboolRaw</i> as Boolean ,
(optional) <i>varResponseObject</i> as Object |
| <u>RenderTime</u> | 1.2 | <i>varDate</i> as Variant ,
<i>varFormat</i> as String ,
(optional) <i>varResponseObject</i> as Object |

Remarks

The ContainerRenderer object can render any subset of the rows of a container object. It can accept a CDO AddressEntries, Folders, Messages, or Recipients collection in its **DataSource** property.

The container object contents rendered by the container renderer are as follows:

| Data source (container object) | Objects (contents) rendered |
|----------------------------------|--|
| <u>AddressEntries</u> collection | <u>AddressEntry</u> |
| <u>Folders</u> collection | <u>Folder</u> |
| <u>Messages</u> collection | <u>AppointmentItem</u> , <u>GroupHeader</u> ,
<u>MeetingItem</u> , <u>Message</u> |

Recipients collection

Recipient

The container renderer inherits all the functionality of the object renderer, and has additional capability specific to rendering an address book container or folder as a table.

You must **Set** the ContainerRenderer object to **Nothing**, or **Set** its DataSource property to **Nothing**, before you call the Session object's Logoff method. Failure to do so can result in unexpected behavior.

BusinessDayEndTime Property (ContainerRenderer Object) Group

The **BusinessDayEndTime** property returns or sets the time of day the business day is set to end. Read/write.

Syntax

objContRend.**BusinessDayEndTime**

Data Type

Variant (**vbDate** format)

Remarks

The **BusinessDayEndTime** property is used when a CalendarView is applied to the container object's CurrentView property. **BusinessDayEndTime** contains only the time portion of a standard date/time field.

The **BusinessDayEndTime** property can be set from the Session object's "BusinessDayEndTime" option. It defaults to 5:00 P.M. (17:00) if not set. The session's options are set by its SetOption method and retrieved with its GetOption method.

The session's "BusinessDayEndTime" option and the container renderer's **BusinessDayEndTime** property are not automatically kept in synchronization. Changing one does not cause the other to be changed. Your application is responsible for setting each of them as appropriate.

BusinessDays Property (ContainerRenderer Object) Group

The **BusinessDays** property returns or sets a bitmask representing the days of the week that are to be considered business days. Read/write.

Syntax

objContRend.**BusinessDays**

Data Type

Long

Remarks

The **BusinessDays** property is used when a CalendarView is applied to the container object's CurrentView property. **BusinessDays** contains a logical **OR** of one or more of the constants used in the RecurrencePattern object's DayOfWeekMask property.

The **BusinessDays** property can be set from the Session object's "WorkingDays" option. It defaults to 62 (**CdoMonday** | ... | **CdoFriday**) if not set. The session's options are set by its SetOption method and retrieved with its GetOption method.

The session's "WorkingDays" option and the container renderer's **BusinessDays** property are not automatically kept in synchronization. Changing one does not cause the other to be changed. Your application is responsible for setting each of them as appropriate.

BusinessDayStartTime Property (ContainerRenderer Object) Group

The **BusinessDayStartTime** property returns or sets the time of day the business day is set to start. Read/write.

Syntax

objContRend.**BusinessDayStartTime**

Data Type

Variant (**vbDate** format)

Remarks

The **BusinessDayStartTime** property is used when a CalendarView is applied to the container object's CurrentView property. **BusinessDayStartTime** contains only the time portion of a standard date/time field. The calendar view scrolls the HTML output to the start of the business day before sending it to the browser.

The **BusinessDayStartTime** property can be set from the Session object's "BusinessDayStartTime" option. It defaults to 9:00 A.M. (09:00) if not set. The session's options are set by its SetOption method and retrieved with its GetOption method.

The session's "BusinessDayStartTime" option and the container renderer's **BusinessDayStartTime** property are not automatically kept in synchronization. Changing one does not cause the other to be changed. Your application is responsible for setting each of them as appropriate.

CellPattern Property (ContainerRenderer Object)

Group

The **CellPattern** property returns or sets a rendering source that determines how every cell in every table row is rendered. Read/write.

Syntax

objContRender.CellPattern

Data Type

String

Remarks

The text to be rendered in each cell is taken directly from the property associated with that cell's column. The contents of the **CellPattern** property specify the rendering for all cells in all rows except for the heading row. The rendering for the heading row cells is specified in the **HeadingCellPattern** property.

The rendering of the table cells can be thought of as being nested. At the outermost level is the cell definition, which encloses the rendering specified by the **CellPattern** property:

```
<TD WIDTH=[determined by CDO]> [CellPattern] </TD>
```

The inner nesting level is controlled by **CellPattern**, which provides an opening font tag for the rendering returned by either the Column object's **RenderUsing** property or the Pattern object's **RenderUsing** property.

The default setting of the **CellPattern** property is

```
<FONT COLOR=000000 SIZE=2>
```

CodePage Property (ContainerRenderer Object) Group

The **CodePage** property returns or sets the code page used by the ContainerRenderer object. Read/write.

Syntax

objContRend.CodePage

Data Type

Variant (Long, Object, or String)

Remarks

If the **CodePage** property is a long integer, it represents the code page to be used for character representation. If **CodePage** is an object, it contains an **IDispatch** pointer to an **IRequest** object. The CDO Rendering Library obtains from this object an HTTP Accept-Language header and sets the code page to the value that most closely matches the header. If **CodePage** is a string, it is treated as an International Standards Organization (ISO) language name, and the code page is set from the Microsoft® Windows NT® registry entry for that language.

Examples of values for the **CodePage** property are "USA" and 1252.

If a long integer value for **CodePage** is invalid, the code page remains unchanged. If a string value is not a recognizable language name, the appropriate default code page for the locale is used. If the requested code page has not been installed in the Windows NT server, an error is returned.

The setting of the **CodePage** property affects character selection and any dependent data considerations. The collating sequence, the sort order, and the formats for time, date, and currency representation are controlled by the **LCID** property, which is automatically set to correspond to the code page information whenever you set **CodePage**.

The settings of **CodePage** and **LCID** do not affect the locale settings of any Session object. Each session has its own SetLocaleIDs method.

CurrentStore Property (ContainerRenderer Object) Group

The **CurrentStore** property sets the message store containing the data source. Write-only.

Syntax

objContRend.**CurrentStore**

Data Type

Object (InfoStore)

Remarks

The **CurrentStore** property supplies the [InfoStore](#) object that holds the container object to be rendered. You only need to set it if you are rendering from public folders on Microsoft® Exchange Server.

Note that not all container objects reside in a message store. An InfoStore object is relevant if the container object is a [Folders](#), [Messages](#), or [Recipients](#) collection, but **CurrentStore** is undefined for an [AddressEntries](#) collection. The container object to be rendered is supplied by the [DataSource](#) property.

Example

This code fragment places the value of the message store containing the [Session](#) object's [Inbox](#) folder into the **CurrentStore** property:

```
' assume valid ContainerRenderer and Session objects
Set objInbox = objSession.Inbox
objContRend.DataSource = objInbox.Messages
strStoreID = objInbox.StoreID
Set objDataSourceStore = objSession.GetInfoStore(strStoreID)
objContRend.CurrentStore = objDataSourceStore
```

CurrentView Property (ContainerRenderer Object)

Group

The **CurrentView** property returns or sets the current view used to render an address book container or folder. Read/write.

Syntax

Set *objView* = *objContRend*.**CurrentView**

Set *objContRend*.**CurrentView** = *objNewView*

objContRend.**CurrentView** = *index*

objContRend.**CurrentView** = *name*

objView

On successful return, contains the [TableView](#) object that is currently applied to the container object being rendered.

objContRend

Required. The ContainerRenderer object.

objNewView

Object. The view that is to become current.

index

Integer. An index into the container renderer's [Views](#) collection ranging from 1 to the collection's **Count** property.

name

String. The display name of an individual [TableView](#) object in the container renderer's Views collection.

Data Type

TableView object

Remarks

The view object must be a view in the container renderer's Views collection. This collection can be accessed with the container renderer's [Views](#) property. If you attempt to set the current view with a view object that cannot be found, for example if the *index* is out of range or there is no view with the specified *name*, the current view remains unchanged.

If you use the *index* parameter, the [TableView](#) object occupying the indicated position in the Views collection becomes the current view.

If you use the *name* parameter, the first view object in the collection having a matching value in its **Name** property becomes the current view.

Setting the **CurrentView** property applies the view to the container object specified in the [DataSource](#) property. If the new view is a table view, its [Columns](#) collection becomes available through the [TableView](#) object's **Columns** property. For more information, see [Rendering Container Objects](#).

Setting the **CurrentView** property also causes the underlying [AddressEntries](#), [Messages](#), or [Recipients](#) collection to be repopulated. This means that the collection is altered to contain only the items passing the new view's restriction, sorted and grouped as specified by the new view. The collection's **Count** property is also refreshed, and access to its members through the **Item** property reflect the new sort order.

Repopulating the collection also means that the filter is inherited from the new view, and the [AddressEntryFilter](#) or [MessageFilter](#) object is revised to reflect the new view's restriction. Any previous

settings of the filter are ignored.

If you change the **CurrentView** property or alter the collection's filter during execution of an indexed loop, the index reverts to 1 in your next iteration of the loop, and the repopulated collection is accessed starting with the first member according to the new restriction and sort specification.

A newly instantiated Views collection always has a default current view. This default can be specified by the directory server or message store underlying the container object. If it is not specified, the CDO Rendering Library sets it to the first view in the Views collection.

DataSource Property (ContainerRenderer Object) Group

The **DataSource** property contains the CDO container object to be rendered. Read/write.

Syntax

objContRend.DataSource

Data Type

Object (AddressEntries collection, Folders collection, Messages collection, or Recipients collection)

Remarks

The **DataSource** property accepts an [AddressEntries](#), [Folders](#), [Messages](#), or [Recipients](#) collection.

Setting the **DataSource** property specifies the container object to be rendered and instantiates a new [Views](#) collection for the container renderer. Any previous folder views are released and replaced from the new Views collection. All common and personal views remain in effect, and any existing custom views are also retained.

If you are rendering an address book container, you must define one or more custom views for it, because it has no predefined common, personal, or folder views. Furthermore, you must **Add** any new custom views to the present [Views](#) collection before you set **DataSource** to an AddressEntries collection. Otherwise the new custom views are not handled properly in the rendering.

Setting the **DataSource** property repopulates the collection underlying the container object and revises its filter, just as setting the container renderer's [CurrentView](#) property does. Any previous settings of the filter are ignored.

If you are preparing to render a folder and set **DataSource** to a Folders or Messages collection, the **CurrentView** property is automatically reset to the new Views collection's default current view. If you are not rendering a folder, **DataSource** and **CurrentView** operate independently, so when you set **DataSource** to an object other than a Folders or Messages collection, **CurrentView** is left unchanged.

For more information, see [Rendering Container Objects](#).

You must **Set** the **DataSource** property to **Nothing**, or **Set** the ContainerRenderer object to **Nothing**, before you call the [Session](#) object's [Logoff](#) method. Failure to do so can result in unexpected behavior.

Example

This code fragment assigns the **DataSource** property to the [Messages](#) collection of the [Session](#) object's [Inbox](#) folder:

```
' assume valid ContainerRenderer and Session objects
Set objInbox = objSession.Inbox
objContRend.DataSource = objInbox.Messages
```

FirstDayOfWeek Property (ContainerRenderer Object)

Group

The **FirstDayOfWeek** property returns or sets the day on which the week is set to start. Read/write.

Syntax

objContRend.**FirstDayOfWeek**

Data Type

Long

Remarks

The **FirstDayOfWeek** property is used when a CalendarView is applied to the container object's CurrentView property. **FirstDayOfWeek** can have exactly one of the following values:

| FirstDayOfWeek setting | Meaning |
|------------------------|--|
| 1 | The calendar week begins on Monday. |
| 2 | The calendar week begins on Tuesday. |
| 3 | The calendar week begins on Wednesday. |
| 4 | The calendar week begins on Thursday. |
| 5 | The calendar week begins on Friday. |
| 6 | The calendar week begins on Saturday. |
| 7 | The calendar week begins on Sunday. |

The **FirstDayOfWeek** property can be set from the Session object's "FirstDayOfWeek" option. It defaults to 7 (Sunday) if not set. The session's options are set by its SetOption method and retrieved with its GetOption method.

The session's "FirstDayOfWeek" option and the container renderer's **FirstDayOfWeek** property are not automatically kept in synchronization. Changing one does not cause the other to be changed. Your application is responsible for setting each of them as appropriate.

Note that the **FirstDayOfWeek** property and the "FirstDayOfWeek" option are compatible with each other but not with the DayOfWeekMask property of the RecurrencePattern object, which uses the mask constants **CdoSunday** through **CdoSaturday**.

Formats Property (ContainerRenderer Object) Group

The **Formats** property returns a single Format object or a Formats collection. Read-only.

Syntax

Set *objFormats* = *objContRend*.**Formats**

Set *objFormat* = *objContRend*.**Formats**(*index*)

Set *objFormat* = *objContRend*.**Formats**(*name*)

objFormats

On successful return, contains the Formats collection of this container renderer.

objContRend

Required. The ContainerRenderer object.

objFormat

On successful return, contains an individual Format object belonging to this container renderer's Formats collection.

index

Integer. An index into the container renderer's Formats collection.

name

String. The reference name of a special-purpose Format object in the collection.

Data Type

Object (Format or Formats collection)

Remarks

Each format in the collection corresponds to a single property, except for special-purpose formats, which do not represent specific properties. Every property to be rendered should be represented by exactly one Format object.

New formats should be added to the collection before the **DataSource** property is set. If you define a new format after changing the data source, it is ignored during the rendering.

If a Format object is to be accessed with the *index* parameter, the value of *index* must be between 1 and the size of the container renderer's Formats collection. This size is available in the collection's **Count** property.

Although the **Formats** property itself is read-only, the collection it returns can be accessed in the normal manner through its **Add** method, and the properties on its member Format objects retain their respective read/write or read-only accessibility.

HeadingCellPattern Property (ContainerRenderer Object)

Group

The **HeadingCellPattern** property returns or sets a rendering source that determines how the heading of each table view column is rendered. Read/write.

Syntax

objContRender.**HeadingCellPattern**

Data Type

String

Remarks

The heading cell is the cell that appears in the heading row above a column in a table view. Its text is specified in the **Name** property of the Column object. The contents of the **HeadingCellPattern** property specify the rendering for the heading cell text. The rendering for all the other cells in the table is specified in the CellPattern property.

The rendering of the heading cells can be thought of as being nested. At the outermost level is the cell definition, which encloses the rendering specified by the **HeadingCellPattern** property:

```
<TD WIDTH=[determined by CDO]> [HeadingCellPattern] </TD>
```

The innermost nesting level is controlled by **HeadingCellPattern**. If it contains a %value% token, the value from the column's **Name** property replaces the token. For example, you could set

HeadingCellPattern to

```
<B>%value%</B>
```

to cause all the heading cells to be rendered in bold text. If **HeadingCellPattern** does not contain a %value% token, the value of the **Name** property is appended to the end of the **HeadingCellPattern** string.

The default setting of the **HeadingCellPattern** property is

```
<FONT COLOR=000000 SIZE=2>%value%</FONT>
```

The *cellPattern* parameter of the RenderHeading method can override the setting of the **HeadingCellPattern** property for a particular rendering.

HeadingRowPrefix Property (ContainerRenderer Object)

Group

The **HeadingRowPrefix** property returns or sets the HTML string to insert at the beginning of a view's heading row. Read/write.

Syntax

objContRender.**HeadingRowPrefix**

Data Type

String

Remarks

The heading row is a single-row table of heading cells each of which serves as a heading for a column in the view. The **HeadingRowPrefix** property can be used to set rendering for certain attributes for the entire heading row. For example, it could contain

```
<TR BGCOLOR="FF0000">
```

to render all the heading cells on a background of red. **HeadingRowPrefix** cannot, however, be used for a global setting of the heading row's attributes.

The default setting of the **HeadingRowPrefix** property is

```
<TR BGCOLOR=CCCC99 VALIGN="TOP">
```

If **HeadingRowPrefix** is changed, it should always contain the <TR> tag.

You can render the heading row each time you output a frame containing table rows, or you can choose to render it only in the first frame.

HeadingRowSuffix Property (ContainerRenderer Object)

Group

The **HeadingRowSuffix** property returns or sets the HTML string to insert at the end of a view's heading row. Read/write.

Syntax

objContRend.**HeadingRowSuffix**

Data Type

String

Remarks

The heading row is a single-row table of heading cells each of which serves as a heading for a column in the view. The **HeadingRowSuffix** property can be used to cancel a heading row rendering. For example, it could contain

```
</B></TR>
```

to restore rendering from bold weight back to normal weight.

The default setting of the **HeadingRowSuffix** property is

```
</TR>
```

If **HeadingRowSuffix** is changed, it should always contain the `</TR>` tag.

You can render the heading row each time you output a frame containing table rows, or you can choose to render it only in the first frame.

Is24HourClock Property (ContainerRenderer Object)

Group

The **Is24HourClock** property indicates whether the calendar is to be rendered in 12-hour or 24-hour mode. Read/write.

Syntax

objContRend.Is24HourClock

Data Type

Boolean

Remarks

The **Is24HourClock** property is used when a CalendarView is applied to the container object's CurrentView property. Set **Is24HourClock** to **True** to render in 24-hour mode, or **False** to render in 12-hour mode.

The **Is24HourClock** property can be set from the Session object's "Is24HourClock" option. It defaults to **False** if not set. The session's options are set by its SetOption method and retrieved with its GetOption method.

The session's "Is24HourClock" option and the container renderer's **Is24HourClock** property are not automatically kept in synchronization. Changing one does not cause the other to be changed. Your application is responsible for setting each of them as appropriate.

LCID Property (ContainerRenderer Object) Group

The **LCID** property returns the locale identifier for the current messaging user. Read-only.

Syntax

objContRend.**LCID**

Data Type

Long

Remarks

A locale is the set of features of a messaging user's environment that are dependent on language, country, culture, and conventions. These features include the character selection, the collating sequence and sort order, and the date, time, and currency formats. The character selection can be changed by setting the **CodePage** property.

A locale identifier (LCID) is a 32-bit value containing a 16-bit language identifier and a 4-bit sort identifier. The Microsoft® Windows NT® macros **SORTIDFROMLCID** and **LANGIDFROMLCID** can be used to extract these identifiers from the LCID.

The **LCID** property is set automatically when you set the **CodePage** property.

The settings of **CodePage** and **LCID** do not affect the locale settings of any Session object. Each session has its own **SetLocaleIDs** method.

LinkPattern Property (ContainerRenderer Object) Group

The **LinkPattern** property returns or sets a rendering source that determines how a link in a table row is rendered. Read/write.

Syntax

objContRend.**LinkPattern**

Data Type

String

Remarks

The **LinkPattern** property supplies rendering information for a link to the object represented in a row of a table view. This link should normally use the complete HTTP syntax.

In a calendar view, a link is rendered from each AppointmentItem object to its related appointment form. The link is rendered on a nonblank property such as Location or Subject, or on StartTime if no other nonblank property can be found on the appointment.

The CDO Rendering Library generates a link for exactly one cell in each row of a table. It attempts to link the cell in the first column that represents a nonempty string property other than the message class. If no such column can be found, the last cell in the row is linked.

The **LinkPattern** property determines the appearance of the link in the HTML output. The following table shows which substitution tokens can be used. Note that their interpretations are not the same as those for either a column's RenderUsing property or a pattern's RenderUsing property.

| Substitution token | Attribute of object being linked |
|--------------------|--|
| %classpath% | A special-purpose format with the name "classpath" for rendering a message object's message class. |
| %obj% | The object's unique identifier, expressed as a hexadecimal string. |
| %rowid% | The object's index in its containing table. |
| %<formatname>% | Any user-defined or system-defined named format. |

Setting **LinkPattern** generates a Format object with the name "message_Link" and adds it to the container renderer's Formats collection. This named format is for internal use only. You should always use the combination of Render and **LinkPattern** to render any link.

PrivateStore Property (ContainerRenderer Object) Group

The **PrivateStore** property sets the InfoStore object to be used to access personal views for a container object. Write-only.

Syntax

objContRend.PrivateStore

Data Type

Object (InfoStore)

Remarks

The **PrivateStore** property represents a messaging user's default message store, which contains that user's personal views. The default message store is the InfoStore containing the user's active Inbox folder.

The **PrivateStore** property is only needed if you plan to do rendering from a Folders or Messages collection. When needed, **PrivateStore** should be set as soon as the container renderer is instantiated, and specifically before the DataSource property is set. If **PrivateStore** is not set at the right time, personal views are not available to the rendering application.

Example

This code fragment determines the user's default message store from the Session object and sets the **PrivateStore** property accordingly:

```
Dim objSess As MAPI.Session
Dim objStore As InfoStore
Dim strStoreID As String
' ... assume objects are valid ...
strStoreID = objSess.Inbox.StoreID
Set objStore = objSess.GetInfoStore(strStoreID)
Set objContRend.PrivateStore = objStore
MsgBox "Your default store is " & objStore.Name
```

Render Method (ContainerRenderer Object) Group

The **Render** method renders the specified rows of a container object.

Syntax

```
strHTML = objContRend.Render(iStyle [, varPageNo] [, varboolRaw] [, varResponseObject] )
```

strHTML

On successful return, contains a string with the HTML hypertext representing the rows of the folder or address book container. However, if the *varResponseObject* parameter is supplied, **Render** returns a value of **Empty**.

objContRend

Required. The ContainerRenderer object.

iStyle

Required. Long. Determines which objects are to be rendered into HTML hypertext. The *iStyle* parameter must match the setting of the **DataSource** property as follows:

| <i>iStyle</i> setting | Value | Data source | Objects rendered |
|----------------------------|-------|---|---|
| CdoFolder Contents | 1 | <u>AddressEntries</u> ,
<u>Messages</u> ,
<u>Recipients</u> | Address entries, group headers and messages, or recipients, but not child folders |
| CdoFolder Hierarchy | 2 | <u>Folders</u> ,
<u>Messages</u> | Child folders, but not address entries, group headers, messages, or recipients |

varpageNo

Optional. Long. The page at which the rendering is to begin. The default value is 1.

varboolRaw

Optional. Boolean. Reserved. Do not use.

varResponseObject

Optional. Object. An Active Server response object used to accumulate HTML output to send to the browser. This parameter is used primarily in ASP applications. If *varResponseObject* is not supplied, the output is written to *strHTML*.

Remarks

The *iStyle* parameter applies to all container objects. You must set it to correspond to the value of the **DataSource** property. If you set it incorrectly, for example to **CdoFolderHierarchy** for a Recipients collection, or if you do not set it, **Render** returns **CdoE_INVALID_PARAMETER**.

Normally only enough rows to fill one frame should be rendered in one call to the **Render** method. The programmer can control this by setting the **RowsPerPage** property. Note that the appropriate number of rows depends on several factors, including the frame size, the font size, what other items are being displayed together with the table rows, and download time to the browser.

The HTML output of the **Render** and **RenderHeading** methods is treated as two separate tables, one for the header row and one for the container contents rows. The **TablePrefix** and **TableSuffix** property strings are inserted around each of these tables.

Example

If the container contains 40 messages, this code fragment processes them in three renderings:

```
Dim pStart As Long ' current starting page
Dim rType As Integer ' type of rows to be rendered
Dim objResp As Object ' Active Server response object
' assume ContainerRenderer object already defined and initialized
objContRend.RowsPerPage = 15 ' 15 rows might fit reasonably in a frame
rType = CdoFolderContents
' ...
pStart = 1
objContRend.Render rType, pStart, , objResp ' pStart = 2 on return
objContRend.Render rType, pStart, , objResp ' pStart = 3 on return
objContRend.Render rType, pStart, , objRes) ' pStart = 3 on return
```

RenderDate Method (ContainerRenderer Object) Group

The **RenderDate** method renders the date portion of the supplied date/time.

Syntax

```
strHTML = objContRend.RenderDate(varDate, varFormat [, varResponseObject] )
```

strHTML

On successful return, contains a string representing the date. However, if the *varResponseObject* parameter is supplied, **RenderDate** returns a value of **Empty**.

objContRend

Required. This ContainerRenderer object.

varDate

Required. Variant (**vbDate** format). The date/time to be rendered as a date.

varFormat

Required. String. The format picture string to use for the date output.

varResponseObject

Optional. Object. An Active Server response object used to accumulate HTML output to send to the browser. This parameter is used primarily in ASP applications. If *varResponseObject* is not supplied, the output is written to *strHTML*.

Remarks

The **RenderDate** method ignores the time component of the **vbDate** format. You can render the time component with the **RenderTime** method.

The *varFormat* parameter specifies a picture for the output. Its contents are defined in the Win32® function **GetDateFormat**.

RenderHeading Method (ContainerRenderer Object)

Group

The **RenderHeading** method renders a table containing the current view's column headers.

Syntax

```
strHTML = objContRend.RenderHeading( [bstrCellPattern] [, varResponseObject] )
```

strHTML

On successful return, contains a string with the HTML hypertext representing the headers. However, if the *varResponseObject* parameter is supplied, **RenderHeading** returns a value of **Empty**.

objContRend

Required. The ContainerRenderer object.

bstrCellPattern

Optional. String. A rendering source specifying the rendering of the heading cell for each column. If supplied, the *bstrCellPattern* parameter overrides the specification in the **HeadingCellPattern** property.

varResponseObject

Optional. Object. An Active Server response object used to accumulate HTML output to send to the browser. This parameter is used primarily in ASP applications. If *varResponseObject* is not supplied, the output is written to *strHTML*.

Remarks

The HTML output of the **Render** and **RenderHeading** methods is treated as two separate tables, one for the header row and one for the container contents rows. The **TablePrefix** and **TableSuffix** property strings are inserted around each of these tables.

You might choose to call **RenderHeading** only in the first frame of your HTML output and then display only item rows in subsequent frames.

RenderProperty Method (ContainerRenderer Object)

Group

The **RenderProperty** method renders the designated property of the parent of the object specified by the **DataSource** property.

Syntax

```
strHTML = objContRend.RenderProperty(varProperty [, varboolRaw] [, varResponseObject] )
```

strHTML

On successful return, contains a string with the HTML hypertext representing the specified property. However, if the *varResponseObject* parameter is supplied, **RenderProperty** returns a value of **Empty**.

objContRend

Required. The ContainerRenderer object.

varProperty

Required. Variant (Long or String). The property tag for the predefined property, or the custom name of the user-defined property, that is to be rendered.

varboolRaw

Optional. Boolean. Reserved. Do not use.

varResponseObject

Optional. Object. An Active Server response object used to accumulate HTML output to send to the browser. This parameter is used primarily in ASP applications. If *varResponseObject* is not supplied, the output is written to *strHTML*.

Remarks

The container object specified by the **DataSource** property does not expose any renderable properties. The **RenderProperty** method renders a property on the parent object of the container object. The correspondence of these objects is as follows:

| Container object in DataSource | Parent object whose properties are rendered by RenderProperty |
|----------------------------------|---|
| <u>AddressEntries</u> collection | <u>AddressList</u> |
| <u>Messages</u> collection | <u>Folder</u> |
| <u>Recipients</u> collection | <u>Message</u> |

The individual properties that can be rendered with the **RenderProperty** method are indicated in the parent object property descriptions.

The *varProperty* parameter designates the property to be rendered. The parameter can be a long integer designating the property by property tag, or a string designating it by custom name. In both cases it corresponds to the **Property** property of the Format object controlling the property to be rendered.

If the *varProperty* parameter is a custom name, it can optionally be prefixed with a GUID string identifying its property set. In this case, the GUID should be enclosed in braces. For more information and examples, see Using Named Properties.

RenderTime Method (ContainerRenderer Object) Group

The **RenderTime** method renders the time portion of the supplied date/time.

Syntax

```
strHTML = objContRend.RenderTime(varDate, varFormat [, varResponseObject] )
```

strHTML

On successful return, contains a string representing the time. However, if the *varResponseObject* parameter is supplied, **RenderDate** returns a value of **Empty**.

objContRend

Required. This ContainerRenderer object.

varDate

Required. Variant (**vbDate** format). The date/time to be rendered as a time.

varFormat

Required. String. The format picture string to use for the time output.

varResponseObject

Optional. Object. An Active Server response object used to accumulate HTML output to send to the browser. This parameter is used primarily in ASP applications. If *varResponseObject* is not supplied, the output is written to *strHTML*.

Remarks

The **RenderTime** method ignores the date component of the **vbDate** format. You can render the date component with the [RenderDate](#) method.

The *varFormat* parameter specifies a picture for the output. Its contents are defined in the Win32® function **GetTimeFormat**.

RowPrefix Property (ContainerRenderer Object) Group

The **RowPrefix** property returns or sets the HTML string to insert at the beginning of the rendering of each row. Read/write.

Syntax

objContRend.RowPrefix

Data Type

String

Remarks

The default setting of the **RowPrefix** property is

```
<TR BGCOLOR=FFFFFF VALIGN="TOP" ALIGN="LEFT">
```

If **RowPrefix** is changed, it should always contain the <TR> tag.

RowsPerPage Property (ContainerRenderer Object) Group

The **RowsPerPage** property returns or sets the number of rows to be rendered with the [Render](#) method. Read/write.

Syntax

objContRend.RowsPerPage

Data Type

Integer

Remarks

The **RowsPerPage** property is set to the number of rows that are to appear together in a single display, that is, the number of rows in each frame of the rendering. This number typically varies between 10 and 25, but could be outside that range under certain conditions. **RowsPerPage** defaults to 25.

Several factors can determine the appropriate number of rows per frame:

- The larger the frame used for the table rendering, the more rows it can hold.
- The smaller the maximum font size used in each row, the more rows can fit in a frame.
- The more rows a frame has, the longer it takes to transmit it to the browser.

If you call the **RenderHeading** method only for the first frame and do not render the heading row in subsequent frames, then you could increase **RowsPerPage** after the first frame.

If your frames include display items such as artwork in addition to the table rows, the space usable for the rows is reduced.

RowSuffix Property (ContainerRenderer Object) Group

The **RowSuffix** property returns or sets the HTML string to insert at the end of the rendering of each row. Read/write.

Syntax

objContRend.RowSuffix

Data Type

String

Remarks

The default setting of the **RowSuffix** property is

```
</TR>
```

If **RowSuffix** is changed, it should always contain the </TR> tag.

TablePrefix Property (ContainerRenderer Object) Group

The **TablePrefix** property returns or sets the HTML string to insert at the beginning of the rendering of a folder or address book container table. Read/write.

Syntax

objContRender.**TablePrefix**

Data Type

String

Remarks

The HTML output of the **Render** and **RenderHeading** methods is treated as two separate tables, one for the header row and one for the item rows. The **TablePrefix** and **TableSuffix** property strings are inserted around each of these tables.

You can set the width of the table to be a percentage of the available horizontal space, or you can use the %tablewidth% substitution token for a fixed value representing the sum of the widths of all the columns. The substitution tokens are described in the **RenderUsing** property of the **Pattern** object.

The default setting of the **TablePrefix** property is

```
<TABLE BORDER=0 CELSPACING=0 CELLPADDING=1 WIDTH=%tablewidth%>  
for a table view and
```

```
<TABLE COLUMNS=%columns% BORDER=0 CELLPADDING=0 CELSPACING=1  
WIDTH=100% HEIGHT=100%>  
for a calendar view.
```

If **TablePrefix** is changed, it should normally contain the <TABLE> tag.

TableSuffix Property (ContainerRenderer Object) Group

The **TableSuffix** property returns or sets the HTML string to insert at the end of the rendering of a folder or address book container table. Read/write.

Syntax

objContRend.**TableSuffix**

Data Type

String

Remarks

The HTML output of the **Render** and **RenderHeading** methods is treated as two separate tables, one for the header row and one for the item rows. The **TablePrefix** and **TableSuffix** property strings are inserted around each of these tables.

The default setting of the **TableSuffix** property is

```
</TABLE>
```

If **TableSuffix** is changed, it should normally contain the </TABLE> tag.

TimeZone Property (ContainerRenderer Object) Group

The **TimeZone** property returns or sets the time zone in which the calendar is to be rendered.
Read/write.

Syntax

objContRend.TimeZone

Data Type

Long

Remarks

The **TimeZone** property is used when a CalendarView is applied to the container renderer's CurrentView property.

The **TimeZone** property can have exactly one of the following values:

| TimeZone setting | Decimal value | Time zone to render calendar in | Difference from GMT |
|------------------------------|---------------|--|---------------------|
| CdoTmzAbuDhabi | 24 | The time zone used in the United Arab Emirates | + 4:00 |
| CdoTmzAdelaide | 19 | The time zone used in central Australia | + 9:30 |
| CdoTmzAlaska | 14 | Alaska time zone (North America) | - 9:00 |
| CdoTmzAlmaty | 46 | The time zone used in Kazakhstan | + 6:00 |
| CdoTmzArizona | 38 | The time zone used in Arizona (USA) | - 7:00 |
| CdoTmzAthens | 7 | The time zone used in Greece | + 2:00 |
| CdoTmzAtlantic Canada | 9 | Atlantic time zone (North America) | - 4:00 |
| CdoTmzAzores | 29 | The time zone used in the Azores | - 1:00 |
| CdoTmzBaghdad | 26 | The time zone used in Iraq | + 3:00 |
| CdoTmzBangkok | 22 | The time zone used in Thailand | + 7:00 |
| CdoTmzBeijing | 45 | The time zone used in mainland China | + 8:00 |
| CdoTmzBerlin | 4 | The time zone used in Germany | + 1:00 |
| CdoTmzBogota | 35 | The time zone used in Colombia | - 5:00 |
| CdoTmzBombay | 23 | The time zone used in India | + 5:30 |

| | | | |
|-----------------------------|----|---|---------|
| CdoTmzBrisbane | 18 | The time zone used in eastern Australia | + 10:00 |
| CdoTmzBuenosAires | 32 | The time zone used in Argentina | - 3:00 |
| CdoTmzCairo | 49 | The time zone used in Egypt | + 2:00 |
| CdoTmzCaracas | 33 | The time zone used in Venezuela | - 4:00 |
| CdoTmzCentral | 11 | Central time zone (North America) | - 6:00 |
| CdoTmzDarwin | 44 | The time zone used in northern Australia | + 9:30 |
| CdoTmzEastern | 10 | Eastern time zone (North America) | - 5:00 |
| CdoTmzEastern Europe | 5 | The time zone used in Latvia, Lithuania, and Romania | + 2:00 |
| CdoTmzEnewetak | 39 | The time zone used on Enewetak | - 12:00 |
| CdoTmzFiji | 40 | The time zone used on the Fijian Islands | + 12:00 |
| CdoTmzGuam | 43 | The time zone used on Guam | + 10:00 |
| CdoTmzGMT | 1 | Greenwich Mean Time, also called UTC (Coordinated Universal Time) | + 0 |
| CdoTmzHarare | 50 | The time zone used in Zimbabwe | + 2:00 |
| CdoTmzHawaii | 15 | Hawaii time zone (North America) | - 10:00 |
| CdoTmzHobart | 42 | The time zone used in Tasmania | + 10:00 |
| CdoTmzHongKong | 21 | The time zone used in Hong Kong SAR, PRC | + 8:00 |
| CdoTmzIndiana | 34 | The time zone used in Indiana (USA) | - 5:00 |
| CdoTmzIslamabad | 47 | The time zone used in Pakistan | + 5:00 |
| CdoTmzIsrael | 27 | The time zone used in Israel | + 2:00 |
| CdoTmzKabul | 48 | The time zone used in Afghanistan | + 4:30 |
| CdoTmzLisbon | 2 | The time zone used in Portugal | + 0 |
| CdoTmzMagadan | 41 | The time zone used in eastern Russia | + 11:00 |
| CdoTmzMax | 52 | The first unused time zone value (not a valid | ----- |

| | | | |
|-----------------------------|----|--|---------|
| | | setting) | |
| CdoTmzMexicoCity | 37 | The time zone used in central and eastern Mexico | - 6:00 |
| CdoTmzMidAtlantic | 30 | The time zone used on the mid-Atlantic islands | - 2:00 |
| CdoTmzMidway Island | 16 | The time zone used on Midway Island | - 11:00 |
| CdoTmzMonrovia | 31 | The time zone used in Liberia | + 0 |
| CdoTmzMoscow | 51 | The time zone used in western Russia | + 3:00 |
| CdoTmzMountain | 12 | Mountain time zone (North America) | - 7:00 |
| CdoTmzNewfoundland | 28 | The time zone used in far eastern Canada | - 3:30 |
| CdoTmzOrigin | 0 | The time zone of the International Date Line, where each calendar day begins (not a valid setting) | + 12:00 |
| CdoTmzPacific | 13 | Pacific time zone (North America) | - 8:00 |
| CdoTmzParis | 3 | The time zone used in France | + 1:00 |
| CdoTmzPrague | 6 | The time zone used in Czechoslovakia | + 1:00 |
| CdoTmzRio de Janeiro | 8 | The time zone used in Brazil | - 3:00 |
| CdoTmzSaskatchewan | 36 | The time zone used in Saskatchewan (Canada) | - 6:00 |
| CdoTmzTehran | 25 | The time zone used in Iran | + 3:30 |
| CdoTmzTokyo | 20 | The time zone used in Japan | + 9:00 |
| CdoTmzWellington | 17 | The time zone used in New Zealand | + 12:00 |

You can also specify whether or not a time zone is susceptible to summer time, also known as daylight saving time. The flag **CdoTmzNoDST**, which has the value &H4000 (= 16384 decimal) indicates that the time zone you are specifying does not change during the year. If you **OR** the time zone identifier with **CdoTmzNoDST**, the difference from GMT is always constant. If you supply the time zone identifier alone, the difference from GMT changes twice a year.

The values **CdoTmzOrigin** and **CdoTmzMax** are for comparison only and are not valid settings. An attempt to set **TimeZone** with a value of **CdoTmzOrigin** or less, or **CdoTmzMax** or greater, returns **CdoE_INVALID_PARAMETER**. The **CdoTmzNoDST** flag is ignored for the purpose of this validity check.

The **TimeZone** property can be set from the [Session](#) object's "TimeZone" option. It defaults to the Web server's current time zone if not set. The session's options are set by its [SetOption](#) method and retrieved with its [GetOption](#) method.

The session's "TimeZone" option and the container renderer's **TimeZone** property are not automatically kept in synchronization. Changing one does not cause the other to be changed. Your application is responsible for setting each of them as appropriate.

Views Property (ContainerRenderer Object) Group

The **Views** property returns a single [CalendarView](#) or [TableView](#) object or a [Views](#) collection object containing all the views on a container object. Read-only.

Syntax

Set *objViewsColl* = *objContRend.Views*

Set *objView* = *objContRend.Views(index)*

objViewsColl

Object. The Views collection of this container renderer.

objContRend

Required. The ContainerRenderer object.

objView

Object. An individual [CalendarView](#) or [TableView](#) object belonging to this container renderer's Views collection.

index

Integer. An index into the container renderer's [Views](#) collection.

Data Type

Object (TableView or Views collection)

Remarks

The [Views](#) collection returned by the **Views** property contains all the predefined common, folder, and personal views on the container object. If you have modified any of these predefined views, your changes are still in effect provided the Views collection has not been released in the meantime. Custom views you have added to the collection are also in effect until the collection is released. Setting the [DataSource](#) property releases the collection, but setting the [CurrentView](#) property does not.

To select the view to be used to render the container object, set the **CurrentView** property to one of the views returned in the **Views** property.

If a view object is to be accessed with the *index* parameter, the value of *index* must be between 1 and the size of the container renderer's [Views](#) collection. This size is available in the collection's [Count](#) property.

Although the **Views** property itself is read-only, the collection it returns can be accessed in the normal manner through its [Add](#) method, and the properties on its member [CalendarView](#) and [TableView](#) objects retain their respective read/write or read-only accessibility.

Format Object

The Format object contains information that controls how a particular property is to be rendered.

At a Glance

| | |
|----------------------------|-------------------------------------|
| Specified in type library: | CDOHTML.DLL |
| First available in: | CDO Rendering Library version 1.1 |
| Parent objects: | Formats collection |
| Child objects: | Patterns collection |
| Default property: | (none) |

Properties

| Name | Available since version | Type | Access |
|--------------------------|-------------------------|--|------------|
| Class | 1.1 | Long | Read-only |
| Name | 1.1 | String | Read/write |
| Parent | 1.1 | Formats collection object | Read-only |
| Patterns | 1.1 | Pattern object or Patterns collection object | Read-only |
| Property | 1.1 | Long or String | Read-only |

Methods

| Name | Available since version | Parameters |
|------------------------|-------------------------|------------|
| Delete | 1.1 | (none) |

Remarks

The Format object provides rendering information for exactly one property of the object being rendered. The property is designated by either property tag or property name in the corresponding Format object's [Property](#) property.

A format contains a collection of patterns that control how all values of the property are to be rendered, available through the format's [Patterns](#) property. Each pattern in the [Patterns](#) collection governs the rendering for a particular set of values of the property.

You can define a format for every renderable property, but it is also possible to render a property without a format. If the property is associated with a column in a table view, that [Column](#) object may have a [RenderUsing](#) property providing a rendering source for the column. If a property is rendered without a format or a rendering source, either in a column or with the [RenderProperty](#) method, it is rendered by data type using default rendering information.

Delete Method (Format Object)

Group

The **Delete** method removes the Format object from the [Formats](#) collection.

Syntax

objFormat.Delete()

objFormat

Required. The Format object.

Remarks

The **Delete** method performs an irreversible operation on the collection. It calls **Release** on the collection's reference to the Format object. If you have another reference to the format, you can still access its properties and methods, but you can never again associate it with any collection because the **Add** method always creates a new object. You should **Set** your reference variable either to **Nothing** or to another format.

The final **Release** on the Format object takes place when you assign your reference variable to **Nothing**, or when you call **Delete** if you had no other reference. At this point the object is removed from memory. Attempted access to a released object results in an error return of **CdoE_INVALID_OBJECT**.

The action of the **Delete** method is permanent, and the Format object cannot be restored to the collection. Before calling **Delete**, your application can prompt the user to verify whether the format should be permanently deleted.

When you delete a member of a collection, the collection is immediately refreshed, meaning that its **Count** property is reduced by one and its members are reindexed. To access a member following the deleted member, you must use its new index value. For more information, see [Looping Through a Collection](#).

Example

This code fragment illustrates the two situations previously explained. The **Set** statement calls **AddRef** on the first Format object. That reference survives the call to **Delete** and has to be reassigned. The second Format object is deleted without creating another reference, and no other action is necessary.

```
' assume valid ObjectRenderer object
Set objFormat = objObjectRenderer.Formats.Item(1)
objFormat.Delete ' still have a reference from Set statement
' ... other operations on objFormat possible but pointless ...
Set objFormat = Nothing ' necessary to remove reference
' ...
objObjectRenderer.Formats.Item(2).Delete ' no reference to remove
```

Name Property (Format Object)

Group

The **Name** property returns or sets the reference name of this Format object. Read/write.

Syntax

objFormat.Name

Data Type

String

Remarks

The **Name** property represents the name used to identify a special-purpose format, which does not represent a specific property.

Patterns Property (Format Object)

Group

The **Patterns** property returns a single Pattern object or a Patterns collection belonging to this format. Read-only.

Syntax

Set *objPatterns* = *objFormat*.**Patterns**

Set *objPattern* = *objFormat*.**Patterns**(*index*)

objPatterns

Object. The Patterns collection of this format.

objFormat

Required. The Format object.

objPattern

Object. An individual Pattern object belonging to this format's Patterns collection.

index

Integer. An index into the format's Patterns collection.

Data Type

Object (Pattern or Patterns collection)

Remarks

Each pattern in the collection specifies rendering for a particular set of values of the property represented by the Format object.

If a Pattern object is to be accessed with the *index* parameter, the value of *index* must be between 1 and the size of the format's Patterns collection. This size is available in the collection's **Count** property.

Although the **Patterns** property itself is read-only, the collection it returns can be accessed in the normal manner through its **Add** method, and the properties on its member Pattern objects retain their respective read/write or read-only accessibility.

Property Property (Format Object)

Group

The **Property** property returns the name or tag of the property to be rendered. Read-only.

Syntax

objFormat.Property

Data Type

Variant (Long or String)

Remarks

The **Property** property is a long integer if the property being rendered is specified by a property tag. If it is a named custom property, the **Property** property is a string. The property name in this string can optionally be prefixed with a GUID string identifying its property set. In this case, the GUID should be enclosed in braces. For more information and examples, see [Using Named Properties](#).

Formats Collection Object

The Formats collection object contains zero or more formats for a rendering.

At a Glance

Specified in type library: CDOHTML.DLL
First available in: CDO Rendering Library version 1.1
Parent objects: [ContainerRenderer](#)
[ObjectRenderer](#)
[RenderingApplication](#)
Child objects: [Format](#)
Default property: [Item](#)

A Formats collection supports count and index values that let you access an individual Format object through the **Item** property. The Formats collection also supports the Microsoft® Visual Basic® **For Each** statement.

Properties

| Name | Available since version | Type | Access |
|------------------------|-------------------------|---|-----------|
| Class | 1.1 | Long | Read-only |
| Count | 1.1 | Long | Read-only |
| Item | 1.1 | Format object | Read-only |
| Parent | 1.1 | ContainerRenderer object or ObjectRenderer object | Read-only |

Methods

| Name | Available since version | Parameters |
|---------------------|-------------------------|--|
| Add | 1.1 | <i>varProperty</i> as Variant ,
(optional) <i>varName</i> as String |

Remarks

The Formats collection object controls how the values of certain properties are rendered. A property represented by a [Format](#) object in the collection is rendered according to the patterns in that format. Every property to be rendered can be represented by at most one Format object.

Add Method (Formats Collection)

Group

The **Add** method creates and returns a new [Format](#) object in the Formats collection.

Syntax

```
Set objFormat = objFormatsColl.Add(varProperty [, varName] )
```

objFormat

On successful return, contains the new Format object.

objFormatsColl

Required. The Formats collection object.

varProperty

Required. Variant (Long or String). The property tag for the predefined property, or the custom name of the user-defined property, that is to be formatted by the new Format object. A value of zero is used to indicate a special-purpose format not representing any property.

varName

Optional. String. The name to be assigned to the new Format object. The *varName* parameter is for special-purpose formats only. If it is specified, the *varProperty* parameter must be set to zero.

Remarks

The **Add** method parameters correspond to the [Property](#) and [Name](#) properties of the [Format](#) object.

The *varProperty* parameter designates the property to be rendered. The parameter can be a long integer designating the property by property tag, or a string designating it by custom name. In both cases it corresponds to the **Property** property.

If the *varProperty* parameter is a custom name, it can optionally be prefixed with a GUID string identifying its property set. In this case, the GUID should be enclosed in braces. For more information and examples, see [Using Named Properties](#).

The *varName* parameter designates a reference name for a special-purpose format, which does not represent a specific property. The format name should not be confused with a property name; in fact, when the *varProperty* parameter supplies a property name, the *varName* parameter must be omitted.

CDO enforces uniqueness of format names within the same Formats collection. If you attempt to set the *varName* parameter to an existing name, **Add** returns an error.

You should avoid giving a format a name that coincides with any of the substitution tokens, as this could cause unexpected results from the rendering methods. For a list of the substitution tokens, see the [RenderUsing](#) property of the [Pattern](#) object.

Every property to be rendered can have at most one format. The Formats collection cannot contain more than one Format object for any one property.

You can add a format corresponding to a predefined property at any time, but if you wish to add a format corresponding to a user-defined property, you must first set the data source of the rendering object you are working with, that is, either the [ContainerRenderer](#) object's [DataSource](#) property or the [ObjectRenderer](#) object's [DataSource](#) property.

If you attempt to add a user-defined property without the data source being set, you get an error return from **Add**. Because the CDO Rendering Library runs primarily in server-side script, which does not support exception handling, you must explicitly test for errors:

```
Set objFormat = objFormatsColl.Add("CustomPropName")
If Err.Number <> 0 Then
    If Err.Number = &H8000FFFF Then ' E_UNEXPECTED - no data source
        ' handle error - perhaps set DataSource property and try again
```

End If
End If

Count Property (Formats Collection)

Group

The **Count** property returns the number of [Format](#) objects in the collection. Read-only.

Syntax

objFormatsColl.Count

Data Type

Long

Remarks

For more information on using the **Count** and **Item** properties, see the example in the [Item](#) property.

Item Property (Formats Collection)

Group

The **Item** property returns the specified Format object from the Formats collection. Read-only.

Syntax

objFormatsColl.**Item**(*index*)

objFormatsColl.**Item**(*name*)

objFormatsColl.**Item**(*propTag*)

index

Integer. Value must be greater than 0 and less than or equal to 65,535 (&HFFFF). Specifies the index within the Formats collection.

name

String. The display name of the Format object to be selected from the collection.

propTag

Long. Value must be greater than or equal to 65,536 (&H10000). Specifies the 32-bit property tag of the renderable property corresponding to a format in the collection. The renderable property is indicated in the Format object's **Property** property.

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

Data Type

Format object

Remarks

The **Item** property works like an accessor property.

If the specified Format object is not found in the collection, the **Item** property returns **Nothing**.

If a format represents a custom property, you cannot select the format by the property name, because the *name* parameter specifies the name of the format itself. In such a case, you must ascertain the property tag for the property and supply it in the *propTag* parameter. You can obtain the property tag of a custom property from the **ID** property of the Field object being used to access the custom property.

Although the **Item** property itself is read-only, the Format object it returns can be accessed in the normal manner, and its properties retain their respective read/write or read-only accessibility.

Example

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

```
' Put all format names in a collection into a string array
Dim strItemName(100) as String
Dim i As Integer ' loop counter
' error handling omitted from this fragment ...
For i = 1 To objFormatsColl.Count Step 1
    strItemName(i) = objFormatsColl.Item(i).Name
    ' or = objFormatsColl(i) since Item and Name are default properties
    If 100 = i Then ' max size of string array
        Exit Function
    End If
Next i
```


ObjectRenderer Object

The ObjectRenderer object renders selected properties of a specified CDO object.

At a Glance

| | |
|----------------------------|--|
| Specified in type library: | CDOHTML.DLL |
| First available in: | CDO Rendering Library version 1.1 |
| Parent objects: | RenderingApplication (or none) |
| Child objects: | Formats collection |
| Default property: | (none) |

Properties

| Name | Available since version | Type | Access |
|-----------------------------|-------------------------|--|------------|
| Class | 1.1 | Long | Read-only |
| CodePage | 1.1 | Long, Object, or String | Read/write |
| DataSource | 1.1 | AddressEntry ,
AppointmentItem ,
Attachment , Folder ,
MeetingItem , or
Message object | Read/write |
| Formats | 1.1 | Format object or
Formats collection
object | Read-only |
| LCID | 1.1 | Long | Read-only |
| LinkPattern | 1.1 | String | Read/write |
| Parent | 1.1 | RenderingApplication
object or set to
Nothing | Read-only |

Methods

| Name | Available since version | Parameters |
|--------------------------------|-------------------------|--|
| RenderDate | 1.2 | <i>varDate</i> as Variant ,
<i>varFormat</i> as String ,
(optional) <i>varResponseObject</i> as
Object |
| RenderLink | 1.1 | (optional) <i>varResponseObject</i> as
Object |
| RenderProperty | 1.1 | <i>varProperty</i> as Variant ,
(optional) <i>varboolRaw</i> as Boolean ,
(optional) <i>varResponseObject</i> as |

| | | |
|--------------------------|-----|---|
| <u>RenderTime</u> | 1.2 | Object
<i>varDate</i> as Variant ,
<i>varFormat</i> as String ,
(optional) <i>varResponseObject</i> as
Object |
|--------------------------|-----|---|

Remarks

The ObjectRenderer object can be applied to a CDO object to render selected properties. For example, you can use it to render the subject properties of a Message object or the subfolders of a Folder object. The object renderer is easier and faster to use than a specialized renderer such as the ContainerRenderer object. You use the object renderer when you need only a few properties and not the full tabular functionality of the specialized rendering object.

You must **Set** the ObjectRenderer object to **Nothing**, or **Set** its DataSource property to **Nothing**, before you call the Session object's **Logoff** method. Failure to do so can result in unexpected behavior.

CodePage Property (ObjectRenderer Object) Group

The **CodePage** property returns or sets the code page used by the ObjectRenderer object. Read/write.

Syntax

objObjectRend.CodePage

Data Type

Variant (Long, Object, or String)

Remarks

If the **CodePage** property is a long integer, it represents the code page to be used for character representation. If **CodePage** is an object, it contains an **IDispatch** pointer to an **IRequest** object. The CDO Rendering Library obtains from this object an HTTP Accept-Language header and sets the code page to the value that most closely matches the header. If **CodePage** is a string, it is treated as an International Standards Organization (ISO) language name, and the code page is set from the Microsoft® Windows NT® registry entry for that language.

Examples of values for the **CodePage** property are "USA" and 1252.

If a long integer value for **CodePage** is invalid, the code page remains unchanged. If a string value is not a recognizable language name, the appropriate default code page for the locale is used. If the requested code page has not been installed in the Windows NT server, an error is returned.

The setting of the **CodePage** property affects character selection and any dependent data considerations. The collating sequence, the sort order, and the formats for time, date, and currency representation are controlled by the **LCID** property, which is automatically set to correspond to the code page information whenever you set **CodePage**.

The settings of **CodePage** and **LCID** do not affect the locale settings of any Session object. Each session has its own SetLocaleIDs method.

DataSource Property (ObjectRenderer Object) Group

The **DataSource** property contains the CDO object for which certain properties are to be rendered. Read/write.

Syntax

objObjectRend.DataSource

Data Type

Object (AddressEntry, AppointmentItem, Attachment, Folder, MeetingItem, or Message)

Remarks

The **DataSource** property accepts an [AddressEntry](#), [AppointmentItem](#), [Attachment](#), [Folder](#), [MeetingItem](#), or [Message](#) object.

You must **Set** the **DataSource** property to **Nothing**, or **Set** the ObjectRenderer object to **Nothing**, before you call the [Session](#) object's [Logoff](#) method. Failure to do so can result in unexpected behavior.

Formats Property (ObjectRenderer Object) Group

The **Formats** property returns a single [Format](#) object or a [Formats](#) collection. Read-only.

Syntax

Set *objFormats* = *objObjectRend*.**Formats**

Set *objFormat* = *objObjectRend*.**Formats**(*index*)

Set *objFormat* = *objObjectRend*.**Formats**(*name*)

objFormats

Object. The Formats collection of this object renderer.

objObjectRend

Required. The ObjectRenderer object.

objFormat

Object. An individual Format object belonging to this object renderer's Formats collection.

index

Integer. An index into the object renderer's [Formats](#) collection.

name

String. The reference name of a special-purpose [Format](#) object in the collection.

Data Type

Object (Format or Formats collection)

Remarks

Each format in the collection corresponds to a single property, except for special-purpose formats, which do not represent specific properties. Every property to be rendered should be represented by exactly one [Format](#) object.

If a Format object is to be accessed with the *index* parameter, the value of *index* must be between 1 and the size of the object renderer's [Formats](#) collection. This size is available in the collection's **Count** property.

Although the **Formats** property itself is read-only, the collection it returns can be accessed in the normal manner through its **Add** method, and the properties on its member [Format](#) objects retain their respective read/write or read-only accessibility.

LCID Property (ObjectRenderer Object)

Group

The **LCID** property returns the locale identifier for the current messaging user. Read-only.

Syntax

objObjectRend.LCID

Data Type

Long

Remarks

A locale is the set of features of a messaging user's environment that are dependent on language, country, culture, and conventions. These features include the character selection, the collating sequence and sort order, and the date, time, and currency formats. The character selection can be changed by setting the **CodePage** property.

A locale identifier (LCID) is a 32-bit value containing a 16-bit language identifier and a 4-bit sort identifier. The Microsoft® Windows NT® macros **SORTIDFROMLCID** and **LANGIDFROMLCID** can be used to extract these identifiers from the LCID.

The **LCID** property is set automatically when you set the **CodePage** property.

The settings of **CodePage** and **LCID** do not affect the locale settings of any Session object. Each session has its own **SetLocaleIDs** method.

LinkPattern Property (ObjectRenderer Object) Group

The **LinkPattern** property returns or sets a rendering source that determines how a link is rendered. Read/write.

Syntax

objObjectRend.LinkPattern

Data Type

String

Remarks

The **LinkPattern** property supplies rendering information for a link to the object currently specified in the **DataSource** property. The link is rendered by the **RenderLink** method. It can use the complete HTTP syntax or simply render a URL.

The **LinkPattern** property determines the appearance of the link in the HTML output. The following table shows which substitution tokens can be used. Note that their interpretations are not the same as for a **Pattern** object's **RenderUsing** property.

| Substitution token | Attribute of object being linked |
|--------------------|--|
| %classpath% | A special-purpose format with the name "classpath" for rendering a message object's message class. |
| %obj% | The object's unique identifier, expressed as a hexadecimal string. |
| %<formatname>% | Any user-defined or system-defined named format. |

Setting **LinkPattern** generates a **Format** object with the name "message_Link" and adds it to the object renderer's **Formats** collection. This named format is for internal use only. You should always use the combination of **RenderLink** and **LinkPattern** to render any link.

RenderDate Method (ObjectRenderer Object) Group

The **RenderDate** method renders the date portion of the supplied date/time.

Syntax

```
strHTML = objObjectRend.RenderDate(varDate, varFormat [, varResponseObject] )
```

strHTML

On successful return, contains a string representing the date. However, if the *varResponseObject* parameter is supplied, **RenderDate** returns a value of **Empty**.

objObjectRend

Required. This ObjectRenderer object.

varDate

Required. Variant (**vbDate** format). The date/time to be rendered as a date.

varFormat

Required. String. The format picture string to use for the date output.

varResponseObject

Optional. Object. An Active Server response object used to accumulate HTML output to send to the browser. This parameter is used primarily in ASP applications. If *varResponseObject* is not supplied, the output is written to *strHTML*.

Remarks

The **RenderDate** method ignores the time component of the **vbDate** format. You can render the time component with the **RenderTime** method.

The *varFormat* parameter specifies a picture for the output. Its contents are defined in the Win32® function **GetDateFormat**.

RenderLink Method (ObjectRenderer Object) Group

The **RenderLink** method renders an HTML link to a specified object.

Syntax

```
strHTML = objObjectRend.RenderLink( [varResponseObject] )
```

strHTML

On successful return, contains a string with the HTML hypertext representing the link. However, if the *varResponseObject* parameter is supplied, **RenderLink** returns a value of **Empty**.

objObjectRend

Required. The ObjectRenderer object.

varResponseObject

Optional. Object. An Active Server response object used to accumulate HTML output to send to the browser. This parameter is used primarily in ASP applications. If *varResponseObject* is not supplied, the output is written to *strHTML*.

Remarks

The **RenderLink** method renders a link to the object currently specified in the **DataSource** property. The rendering information is supplied in the **LinkPattern** property.

RenderProperty Method (ObjectRenderer Object)

Group

The **RenderProperty** method renders the designated property of the object specified by the **DataSource** property.

Syntax

```
strHTML = objObjectRend.RenderProperty(varProperty [, varboolRaw] [, varResponseObject] )
```

strHTML

On successful return, contains a string with the HTML hypertext representing the specified property. However, if the *varResponseObject* parameter is supplied, **RenderProperty** returns a value of **Empty**.

objObjectRend

Required. The ObjectRenderer object.

varProperty

Required. Variant (Long or String). The property tag for the predefined property, or the custom name of the user-defined property, that is to be rendered.

varboolRaw

Optional. Boolean. Reserved. Do not use.

varResponseObject

Optional. Object. An Active Server response object used to accumulate HTML output to send to the browser. This parameter is used primarily in ASP applications. If *varResponseObject* is not supplied, the output is written to *strHTML*.

Remarks

The individual properties that can be rendered with the **RenderProperty** method are indicated in the appropriate object property descriptions.

The *varProperty* parameter designates the property to be rendered. The parameter can be a long integer designating the property by property tag, or a string designating it by custom name. In both cases it corresponds to the **Property** property of the **Format** object controlling the property to be rendered.

If the *varProperty* parameter is a custom name, it can optionally be prefixed with a GUID string identifying its property set. In this case, the GUID should be enclosed in braces. For more information and examples, see [Using Named Properties](#).

RenderTime Method (ObjectRenderer Object) Group

The **RenderTime** method renders the time portion of the supplied date/time.

Syntax

```
strHTML = objObjectRend.RenderTime(varDate, varFormat [, varResponseObject] )
```

strHTML

On successful return, contains a string representing the time. However, if the *varResponseObject* parameter is supplied, **RenderTime** returns a value of **Empty**.

objObjectRend

Required. This ObjectRenderer object.

varDate

Required. Variant (**vbDate** format). The date/time to be rendered as a time.

varFormat

Required. String. The format picture string to use for the time output.

varResponseObject

Optional. Object. An Active Server response object used to accumulate HTML output to send to the browser. This parameter is used primarily in ASP applications. If *varResponseObject* is not supplied, the output is written to *strHTML*.

Remarks

The **RenderTime** method ignores the date component of the **vbDate** format. You can render the date component with the [RenderDate](#) method.

The *varFormat* parameter specifies a picture for the output. Its contents are defined in the Win32® function **GetTimeFormat**.

Pattern Object

The Pattern object represents a rendering pattern within a format.

At a Glance

| | |
|----------------------------|-------------------------------------|
| Specified in type library: | CDOHTML.DLL |
| First available in: | CDO Rendering Library version 1.1 |
| Parent objects: | Patterns collection |
| Child objects: | (none) |
| Default property: | (none) |

Properties

| Name | Available since version | Type | Access |
|-----------------------------|-------------------------|----------------------------|------------|
| Class | 1.1 | Long | Read-only |
| Parent | 1.1 | Patterns collection object | Read-only |
| RenderUsing | 1.1 | String | Read/write |
| Value | 1.1 | Variant | Read/write |

Methods

| Name | Available since version | Parameters |
|------------------------|-------------------------|------------|
| Delete | 1.1 | (none) |

Remarks

The Pattern object governs how a particular set of values of a property are to be rendered. The property is specified by the **Property** property of the [Format](#) object owning the parent [Patterns](#) collection. Each Pattern object in the collection specifies the rendering for the property when its value matches the contents of the pattern's **Value** property.

Delete Method (Pattern Object)

Group

The **Delete** method removes the Pattern object from the [Patterns](#) collection.

Syntax

objPattern.Delete()

objPattern

Required. The Pattern object.

Remarks

The **Delete** method performs an irreversible operation on the collection. It calls **Release** on the collection's reference to the Pattern object. If you have another reference to the pattern, you can still access its properties and methods, but you can never again associate it with any collection because the **Add** method always creates a new object. You should **Set** your reference variable either to **Nothing** or to another pattern.

The final **Release** on the Pattern object takes place when you assign your reference variable to **Nothing**, or when you call **Delete** if you had no other reference. At this point the object is removed from memory. Attempted access to a released object results in an error return of **CdoE_INVALID_OBJECT**.

The action of the **Delete** method is permanent, and the Pattern object cannot be restored to the collection. Before calling **Delete**, your application can prompt the user to verify whether the pattern should be permanently deleted.

When you delete a member of a collection, the collection is immediately refreshed, meaning that its **Count** property is reduced by one and its members are reindexed. To access a member following the deleted member, you must use its new index value. For more information, see [Looping Through a Collection](#).

Example

This code fragment illustrates the two situations previously explained. The **Set** statement calls **AddRef** on the first Pattern object. That reference survives the call to **Delete** and has to be reassigned. The second Pattern object is deleted without creating another reference, and no other action is necessary.

```
' assume valid Format object
Set objPattern = objFormat.Patterns.Item(1)
objPattern.Delete ' still have a reference from Set statement
' ... other operations on objPattern possible but pointless ...
Set objPattern = Nothing ' necessary to remove reference
' ...
objFormat.Patterns.Item(2).Delete ' no reference to remove
```

RenderUsing Property (Pattern Object)

Group

The **RenderUsing** property returns or sets a rendering source that determines how a particular value set of a designated property is rendered. Read/write.

Syntax

objPattern.RenderUsing

Data Type

Variant (String)

Remarks

The **RenderUsing** property provides a source for rendering a property into HTML hypertext. The property to be rendered is designated in the parent Format object's **Property** property. If that property has the value or values specified in this pattern's **Value** property, this **RenderUsing** string is used to render it. Otherwise, the rendering uses a pattern with a matching **Value** property.

If the renderable property is a column property in a table view, the rendering information is taken from the Column object's **RenderUsing** property. If no column rendering information is available, the rendering object searches for a format representing the renderable property. If no such format can be found, or if the format contains no pattern with a value match, the property is rendered by default according to its data type and value.

If the **RenderUsing** string contains substitution tokens within percent signs, such as %value%, the tokens are replaced by the appropriate attributes of the designated property to generate the HTML hypertext. If there are no substitution tokens in the string, the string itself is rendered without modification.

This approach lets you keep compact values in a property, such as enumerated integers instead of lengthier strings. You can then use the integer value as an index into the Patterns collection through each pattern's **Value** property. The pattern's **RenderUsing** property can contain the rendering string appropriate for that particular integer value.

The substitution tokens that can be inserted in a rendering source are as follows:

Substitution

| token | Attribute of property being rendered |
|-------------------|--|
| %apptlength%
% | For a property on an <u>AppointmentItem</u> object being viewed in CdoModeCalendarDaily mode, the number of rows spanned by the appointment or free block. |
| %apptwidth%
% | For a property on an <u>AppointmentItem</u> object being viewed in CdoModeCalendarDaily mode, the number of columns spanned by the appointment or free block. |
| %classpath% | For a property on a <u>Message</u> object, the message class expressed as a lowercase string, such as ipm.note. For a report message class, only the first and last elements are retained, so that REPORT.IPM.Note.NDR is expressed as report.ndr. When the application sets the FormsRoot property of the <u>RenderingApplication</u> object, a special-purpose format named %classpath% is created, which further processes the output of this token. |
| %columns% | For a property on an <u>AppointmentItem</u> object being viewed |

in **CdoModeCalendarDaily** mode, the total number of columns in the view.

- %date%** For a property on an AppointmentItem object, the day for which appointments are being rendered, expressed as a string.
- %kvalue%** For a numeric property, its value expressed in kilobytes, that is, the value divided by 1024. This value is rendered without a "K" character.
- %obj%** For a property on any object, the unique identifier of the object, expressed as a hexadecimal string.
- %parentobj%** For a property on a Message object, the unique identifier of the parent folder of the message, expressed as a hexadecimal string.
- %rowid%** For a property on an object in a calendar view or a table view, the position of the object in its containing table. This position is also the row number in the view.
- %tablewidth%** For a property on an object in a calendar view, the sum of the pixel widths of all the columns in the view, expressed in pixels. The **%tablewidth%** token is primarily useful for the **TablePrefix** property of the ContainerRenderer object.
- %time%** For a property on an AppointmentItem object being viewed in **CdoModeCalendarDaily** mode, the time of the time slot currently being rendered, expressed as a string. If the slot begins on an hour boundary, the string contains the hour and either "AM" or "PM". Otherwise, the string contains the time separator character and the starting minute.
- %value%** For a property on any object, the value of the property, rendered according to the property's data type. In the context of the **CellPattern** or **HeadingCellPattern** property of the ContainerRenderer object, the string returned by the Pattern object's **RenderUsing** property or the Column object's **RenderUsing** property.

A substitution token can also contain the name of a Format object between the percent signs. If the token matches a format's **Name** property, the patterns of that format are searched to find the **RenderUsing** string. This approach lets you bypass the predefined tokens and customize your rendering.

If the character string between the percent signs is not a valid substitution token or format name, the property is rendered by data type. If a token is valid but does not apply to the property being rendered, for example a **%kvalue%** token for a string property, nothing is rendered in place of the token.

More than one substitution token can be included in a **RenderUsing** string, for example **%value%**, **%obj%**, and **%parentobj%**.

The **%classpath%** token is rendered from the MAPI property PR_MESSAGE_CLASS. The **%obj%** token is rendered from a long-term entry identifier if one is available in the underlying table, and otherwise from the MAPI property PR_ENTRYID. The **%parentobj%** token is rendered from the MAPI property PR_PARENT_ENTRYID, and the **%rowid%** token from the MAPI property PR_ROWID.

The PR_MESSAGE_CLASS property can contain characters that are illegal in file names and URLs,

namely blanks, double quotes, and any of the characters in the string "<>[]*.;'/|?@=&%#". If CDO encounters any such character when rendering %classpath%, it converts it to an underscore followed by a two-character string representing the hexadecimal value of the original character. This is particularly important when CDO is rendering forms.

Example

The renderable property is specified in a column's **Property** property, or a format's **Property** property if the column does not supply rendering information.

If the renderable property contains a string:

```
www.microsoft.com
```

and the **RenderUsing** string contains a substitution token:

```
<A HREF=http://%value%>Microsoft Corporation</A>
```

it is rendered with the token replaced by the appropriate attribute:

```
<A HREF=http://www.microsoft.com>Microsoft Corporation</A>
```

If the **RenderUsing** string contains no substitution token:

```
<A HREF=http://msft.com>Microsoft Corporation</A>
```

it is rendered unchanged:

```
<A HREF=http://msft.com>Microsoft Corporation</A>
```

If the **RenderUsing** string contains an invalid or misspelled token:

```
<A HREF=http://%valu%>Microsoft Corporation</A>
```

it is rendered with the token unchanged:

```
<A HREF=http://%valu%>Microsoft Corporation</A>
```

If the **RenderUsing** property contains an inapplicable token:

```
<A HREF=http://%kvalue%>Microsoft Corporation</A>
```

it is rendered without any output for the token:

```
<A HREF=http://>Microsoft Corporation</A>
```


Value Property (Pattern Object)

Group

The **Value** property indicates which property value or values are to be rendered using this pattern. Read/write.

Syntax

objPattern.Value

Data Type

Variant

Remarks

The **Value** property specifies a set of values for the property designated in the Format object's Property property. If the designated property contains a value within this set of values, it is rendered as specified in the RenderUsing property.

You should set the **Value** property with the same data type as that of the format's designated property. However, a string consisting of a single asterisk (*) can be used to match all possible values of the designated property, no matter what its data type is.

If the designated property is a string, the **Value** property can have the following regular expressions embedded within it:

Regular

expression

Matching values

| | |
|---------|--|
| * | Zero or more characters of any values. |
| ? | Exactly one character of any value. |
| [...] | A single character matching any character within the brackets. |
| [x1-x2] | A single character matching any character in the range between the characters x1 and x2 inclusive. |

If the data type of the designated property is PT_BINARY, a valid hexadecimal string in the **Value** property is converted to a binary value for purposes of matching.

If no pattern can be found with a value match, or if no format has been defined for the renderable property, it is rendered by default according to its data type and value.

Common values of the **Value** property include:

```
"*"
True
CdoLow
CdoCc
CdoFileLink
"*Urgent*"
"\\PAYROLL\ARCHIVE\198?\B.XLS"
"All[ae]n H. Anders[eo]n"
"\\SERVER[1-5]\PRODUCTS"
```

The backslash character introduces an escape sequence in regular expression parsing, so you must use two consecutive backslashes to represent a single backslash.

Note that type library constants such as **True**, **CdoLow**, **CdoCc**, and **CdoFileLink** are not available in VBScript.

Patterns Collection Object

The Patterns collection object contains zero or more patterns in a format.

At a Glance

Specified in type library: CDOHTML.DLL
First available in: CDO Rendering Library version 1.1
Parent objects: [Format](#)
Child objects: [Pattern](#)
Default property: [Item](#)

A Patterns collection supports count and index values that let you access an individual Pattern object through the **Item** property. The Patterns collection also supports the Microsoft® Visual Basic® **For Each** statement.

Properties

| Name | Available since version | Type | Access |
|------------------------|-------------------------|----------------|-----------|
| Class | 1.1 | Long | Read-only |
| Count | 1.1 | Long | Read-only |
| Item | 1.1 | Pattern object | Read-only |
| Parent | 1.1 | Format object | Read-only |

Methods

| Name | Available since version | Parameters |
|---------------------|-------------------------|--|
| Add | 1.1 | <i>varValue</i> as Variant ,
<i>varRenderUsing</i> as Variant |

Remarks

The parent [Format](#) object represents a property to be rendered. The Patterns collection should contain enough patterns to cover all possible values of this property. Each [Pattern](#) object's **Value** property indicates the set of values to be rendered by that pattern. If only one Pattern object is included in the collection, its **Value** property should contain a string value consisting of a single asterisk (*), which matches all possible values of the property to be rendered.

Add Method (Patterns Collection)

Group

The **Add** method creates and returns a new Pattern object in the Patterns collection.

Syntax

Set *objPattern* = *objPatternsColl*.**Add**(*varValue*, *varRenderUsing*)

objPattern

On successful return, contains the new Pattern object.

objPatternsColl

Required. The Patterns collection object.

varValue

Required. Variant. Specifies which property values are to be rendered using the new Pattern object.

varRenderUsing

Required. Variant. Specifies how a property is to be rendered using the new Pattern object.

Remarks

The **Add** method parameters correspond to the Value and RenderUsing properties of the new Pattern object.

Count Property (Patterns Collection)

Group

The **Count** property returns the number of [Pattern](#) objects in the collection. Read-only.

Syntax

objPatternsColl.Count

Data Type

Long

Remarks

For more information on using the **Count** and **Item** properties, see the example in the [Item](#) property.

Item Property (Patterns Collection)

Group

The **Item** property returns the specified Pattern object from the Patterns collection. Read-only.

Syntax

objPatternsColl.**Item**(*index*)

index

A short integer ranging from 1 to *objPatternsColl*.**Count**.

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

Data Type

Pattern object

Remarks

The **Item** property works like an accessor property.

If the specified Pattern object is not found in the collection, the **Item** property returns **Nothing**.

Although the **Item** property itself is read-only, the Pattern object it returns can be accessed in the normal manner, and its properties retain their respective read/write or read-only accessibility.

Example

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

```
' Put all pattern values in a collection into a string array
Dim strItemValue(100) As String
Dim i As Integer ' loop counter
' error handling omitted from this fragment ...
For i = 1 To objPatternsColl.Count Step 1
    strItemValue(i) = objPatternsColl.Item(i).Value
    ' or = objPatternsColl(i).Value since Item is default property
    If 100 = i Then ' max size of string array
        Exit Function
    End If
Next i
```

RenderingApplication Object

The RenderingApplication object provides a framework and support for specific rendering objects.

At a Glance

| | |
|----------------------------|--|
| Specified in type library: | CDOHTML.DLL |
| First available in: | CDO Rendering Library version 1.1 |
| Parent objects: | (none) |
| Child objects: | <u>ContainerRenderer</u>
<u>Formats</u> collection
<u>ObjectRenderer</u> |
| Default property: | (none) |

Properties

| Name | Available since version | Type | Access |
|------------------------|-------------------------|-------------------------------|------------|
| <u>Class</u> | 1.1 | Long | Read-only |
| <u>CodePage</u> | 1.1 | Long, Object, or String | Read/write |
| <u>ConfigParameter</u> | 1.1 | Variant | Read-only |
| <u>Formats</u> | 1.1 | Formats collection object | Read-only |
| <u>FormsRoot</u> | 1.1 | String | Read/write |
| <u>ImpID</u> | 1.2 | Long | Read-only |
| <u>LCID</u> | 1.1 | Long | Read-only |
| <u>LoggingLevel</u> | 1.1 | Long | Read/write |
| <u>Name</u> | 1.1 | String | Read/write |
| <u>Parent</u> | 1.1 | Object; set to Nothing | Read-only |
| <u>Version</u> | 1.1 | String | Read-only |
| <u>VirtualRoot</u> | 1.1 | String | Read/write |

Methods

| Name | Available since version | Parameters |
|-------------------------|-------------------------|--|
| <u>CreateRenderer</u> | 1.1 | <i>iClass</i> as Integer |
| <u>Impersonate</u> | 1.2 | <i>dwImpID</i> as Long |
| <u>ListComposeForms</u> | 1.2 | <i>bstrLinkPattern</i> as String ,
(optional) <i>varLanguageDirectory</i> as |

LoadConfiguration

1.1

String

eSource as **Integer**,
bstrSection as **String**,
(optional) *varSession* as **Object**

Remarks

The RenderingApplication object provides a framework for a rendering application. You can set options on the RenderingApplication object that are inherited by all rendering objects created by the **CreateRenderer** method. The interface instantiated by the RenderingApplication object also provides for event logging and performance monitoring.

A RenderingApplication object is considered a top-level object, meaning it can be created directly from a Microsoft® Visual Basic® program. In the CDO Rendering Library it has a ProgID of AMHTML.Application. This code fragment creates a RenderingApplication object through early binding:

```
Dim objRendApp As RenderingApplication  
Set objRendApp = CreateObject ("AMHTML.Application")
```

This code fragment creates a RenderingApplication object through late binding:

```
Dim objRendApp As Object  
Set objRendApp = CreateObject ("AMHTML.Application")
```

Generally, early binding is preferable, because it enforces type checking and generates more efficient code. Note that you specify the full ProgID "AMHTML.Application" instead of just "Application" in order to distinguish a CDO Rendering application from other types of applications available to a Visual Basic program through other object libraries.

CodePage Property (RenderingApplication Object) Group

The **CodePage** property returns or sets the code page to be used by all rendering objects created with the **CreateRenderer** method. Read/write.

Syntax

objRendApp.CodePage

Data Type

Variant (Long, Object, or String)

Remarks

If the **CodePage** property is a long integer, it represents the code page to be used for character representation. If **CodePage** is an object, it contains an **IDispatch** pointer to an **IRequest** object. The CDO Rendering Library obtains from this object an HTTP Accept-Language header and sets the code page to the value that most closely matches the header. If **CodePage** is a string, it is treated as an International Standards Organization (ISO) language name, and the code page is set from the Microsoft® Windows NT® registry entry for that language.

Examples of values for the **CodePage** property are "USA" and 1252.

If a long integer value for **CodePage** is invalid, the code page remains unchanged. If a string value is not a recognizable language name, the appropriate default code page for the locale is used. If the requested code page has not been installed in the Windows NT server, an error is returned.

The setting of the **CodePage** property affects character selection and any dependent data considerations. The collating sequence, the sort order, and the formats for time, date, and currency representation are controlled by the **LCID** property, which is automatically set to correspond to the code page information whenever you set **CodePage**.

The settings of **CodePage** and **LCID** do not affect the locale settings of any **Session** object. Each session has its own **SetLocaleIDs** method.

ConfigParameter Property (RenderingApplication Object)

Group

The **ConfigParameter** property returns the value of a named configuration parameter. Read-only.

Syntax

objRenderApp.**ConfigParameter**(*parameter*)

parameter

Required. String. The name of the parameter read from a configuration section.

Data Type

Variant

Remarks

The **ConfigParameter** property accesses the values of the named configuration parameters read in by the **LoadConfiguration** method. Each parameter is loaded into either the Microsoft® Exchange Directory Server (DS) or the Microsoft® Windows® registry.

The following table lists each parameter and its corresponding user interface setting in the HTTP Protocol property sheets:

| Configuration parameter | Data type | HTTP Protocol property sheet setting |
|-------------------------------|----------------------------|--|
| "Admin Display Name" | String | (DS) Display name |
| "Admin Note" | String | (DS) Administrative note |
| "Anonymous Access" | Boolean | (DS) Allow anonymous users to access the public folders |
| "AnonymousSessionTimeout" | Long | (Registry) Session timeout for anonymous users (default 20 minutes) |
| "AuthenticatedSessionTimeout" | Long | (Registry) Session timeout for authenticated users (default 60 minutes) |
| "Debug" | Long
used as
Boolean | (Registry) If nonzero, append more details to error messages sent to the browser (defaults to 0) |
| "Directory Name" | String | (DS) Directory name |
| "Enterprise" | String | (Registry) Organization name (corresponding to X.400 "/o=") |
| "HTTP Enabled" | Long
used as
Boolean | (DS) Enable HTTP protocol service |
| "Language Pack Directory" | String | (Registry) Physical directory location of langpack DLLs |
| "Publish GAL" | Boolean | (DS) Allow anonymous users to browse the global address list |

| | | |
|-----------------------------|----------------------|--|
| "Publish GAL Limit" | Long | (DS) Maximum number of entries (Advanced page) |
| "Published Public Folders" | String array | (DS) Public folder entry identifiers added to Folder Shortcuts page of Microsoft Exchange Server Administrator utility |
| "RFC1867NoCleanupAt Unload" | Long used as Boolean | (Registry) Do not attempt to delete files in the temporary Web server directory at shutdown (defaults to 0) |
| "RFC1867SaveDirectory" | String | (Registry) Path to a temporary directory on the Web server to hold attachments to messages still being composed |
| "RFC1867Trace" | Long used as Boolean | (Registry) Log all file uploads to a TRCnnnnn.TMP file in the temporary Web server directory (defaults to 0) |
| "Server" | String | (Registry) Common Name (corresponding to X.400 "/cn=") |
| "Site" | String | (Registry) Organizational Unit name (corresponding to X.400 "/ou=") |

The parameters in the registry can be accessed using the registry key

HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\MSExchangeWEB\Parameters

The parameters in the DS can be accessed using the property page

Exchange Administrator\Site name\Configuration\Protocol\HTTP(Web) Site Setting Properties

in the Microsoft Exchange Server Administrator utility.

The "Enterprise", "Site", and "Server" parameters are used together to form the distinguished name of the Microsoft Exchange Server. If these three parameters are incorrect or missing from the registry, the DS based configuration cannot be read and the service cannot run.

You can also use **ConfigParameter** to set and retrieve registry values that are unrelated to CDO Rendering. If you do this, such values are ignored by the CDO Rendering Library.

CreateRenderer Method (RenderingApplication Object)

Group

The **CreateRenderer** method creates a rendering object attached to the rendering application.

Syntax

Set *objRenderer* = *objRendApp*.**CreateRenderer**(*iClass*)

objRenderer

On successful return, contains the new rendering object.

objRendApp

Required. The RenderingApplication object.

iClass

Required. Integer. The class of rendering object to create. The *iClass* parameter can have exactly one of the following values:

| <i>iClass</i>
setting | Decimal
value | Meaning |
|--|------------------|---|
| CdoClass
ContainerRender
er | 3 | Create a <u>ContainerRenderer</u> object. |
| CdoClass
ObjectRenderer | 2 | Create an <u>ObjectRenderer</u> object. |

Remarks

The rendering object created by the **CreateRenderer** method inherits the code page and formats from this application, as well as logging capability. You could create a container renderer or object renderer directly by calling the Microsoft® Visual Basic® **CreateObject** function, but you would sacrifice the inheritance.

Formats Property (RenderingApplication Object) Group

The **Formats** property returns a single [Format](#) object or a [Formats](#) collection. Read-only.

Syntax

Set *objFormats* = *objRenderApp*.**Formats**

Set *objFormat* = *objRenderApp*.**Formats**(*index*)

Set *objFormat* = *objRenderApp*.**Formats**(*name*)

objFormats

Object. The Formats collection of this rendering application.

objRenderApp

Required. The RenderingApplication object.

objFormat

Object. An individual Format object belonging to this rendering application's Formats collection.

index

Integer. An index into the rendering application's [Formats](#) collection.

name

String. The reference name of a special-purpose [Format](#) object in the collection.

Data Type

Object (Format or Formats collection)

Remarks

Each format in the collection corresponds to a single property, except for special-purpose formats, which do not represent specific properties. Every property to be rendered can be represented by at most one [Format](#) object.

The collection of rendering formats returned by the **Formats** property is inherited by all rendering objects created by the [CreateRenderer](#) method. Each format corresponds to one property to be rendered.

If a Format object is to be accessed with the *index* parameter, the value of *index* must be between 1 and the size of the rendering application's [Formats](#) collection. This size is available in the collection's [Count](#) property.

Although the **Formats** property itself is read-only, the collection it returns can be accessed in the normal manner through its [Add](#) method, and the properties on its member [Format](#) objects retain their respective read/write or read-only accessibility.

FormsRoot Property (RenderingApplication Object)

Group

The **FormsRoot** property returns or sets the absolute path to a forms directory tree on Microsoft® Internet Information Server (IIS). Read/write.

Syntax

objRendApp.FormsRoot

Data Type

String

Remarks

The **FormsRoot** property contains the complete path to an IIS disk directory used as the root of a tree of subdirectories containing .ASP files. Each of these subdirectories corresponds to a message class which can be rendered into HTML hypertext. The tree is considered to have its root at a path node ending with the string "\Forms". If the path specified in **FormsRoot** does not end with this string, it is appended to **FormsRoot** to access the tree.

Setting the **FormsRoot** property causes the rendering application to generate a special-purpose format with the name "classpath". This format contains a pattern for each subdirectory in the tree that contains at least one .ASP file. The %classpath% format is used to process the output of the %classpath% substitution token when it appears in the **RenderUsing** property of a Pattern object.

Note that the **FormsRoot** property contains an absolute disk directory path at IIS, while the **VirtualRoot** property contains an HTTP path at the browser. **FormsRoot** must be set to a full physical disk directory path, not a URL or a URL fragment. A path using UNC is acceptable for **FormsRoot**.

If the **FormsRoot** property has not been set, the %classpath% format is not defined, and the output of the %classpath% substitution token in the pattern's **RenderUsing** property is used without modification. You must set **FormsRoot** to the appropriate path if %classpath% is to be used to access a disk directory.

The MAPI property PR_MESSAGE_CLASS, which CDO uses to render the %classpath% token, can contain characters that are illegal in file names and URLs, namely blanks, double quotes, and any of the characters in the string "<>[]*.;'/!/?@=&%#". If CDO encounters any such character when rendering %classpath%, it converts it to an underscore followed by a two-character string representing the hexadecimal value of the original character. This is particularly important when CDO is rendering forms.

Example

Assuming the following IIS disk directory structure:

```
C:\exchsrvr\webdata\usa\forms
  \ipm
    \note*.asp
    \post*.asp
    \document*.asp
  \report
    \dr [with no .ASP files]
    \ndr*.asp
    \ipnm*.asp
```

the %classpath% format is generated to contain the following patterns:

Value property

RenderUsing property

*

ipm.post*

ipm.document*

report.ndr*

report.ipnrm*

ipm/note/

ipm/post/

ipm/document/

report/ndr/

report/ipnrm/

Impersonate Method (RenderingApplication Object) Group

The **Impersonate** method uses a saved security context handle to impersonate an authenticated messaging user.

Syntax

objRenderApp.**Impersonate**(*dwImpID*)

objRenderApp

Required. The RenderingApplication object.

dwImpID

Required. Long. The saved security context handle, or zero to revert to unauthenticated access.

Remarks

The handle to the Microsoft® Windows NT® security context can be obtained from the **ImpID** property and saved in the Session object while the authenticated messaging user is logged on to the Microsoft® Exchange Server. For more information, see Impersonation.

ImpID Property (RenderingApplication Object) Group

The **ImpID** property returns the security context handle for the current messaging user. Read-only.

Syntax

objRenderApp.ImpID

Data Type

Long

Remarks

The handle points to the Microsoft® Windows NT® security context that permits the current messaging user to make authenticated access to the Microsoft® Exchange Server, for example to open a mailbox. The security context can be saved in the [Session](#) object and used later by the [Impersonate](#) method to impersonate the current messaging user on an unauthenticated thread. For more information, see [Impersonation](#).

LCID Property (RenderingApplication Object) Group

The **LCID** property returns the locale identifier for the current messaging user. Read-only.

Syntax

objRenderApp.LCID

Data Type

Long

Remarks

A locale is the set of features of a messaging user's environment that are dependent on language, country, culture, and conventions. These features include the character selection, the collating sequence and sort order, and the date, time, and currency formats. The character selection can be changed by setting the **CodePage** property.

A locale identifier (LCID) is a 32-bit value containing a 16-bit language identifier and a 4-bit sort identifier. The Microsoft® Windows NT® macros **SORTIDFROMLCID** and **LANGIDFROMLCID** can be used to extract these identifiers from the LCID.

The **LCID** property is set automatically when you set the **CodePage** property.

The settings of **CodePage** and **LCID** do not affect the locale settings of any Session object. Each session has its own **SetLocaleIDs** method.

ListComposeForms Method (RenderingApplication Object)

Group

The **ListComposeForms** method returns an array of compose form strings.

Syntax

```
strArrayFormList = objRendApp.ListComposeForms(bstrLinkPattern, varLanguageDirectory)
```

strArrayFormList

On successful return, contains a string array of designated compose form strings.

objRendApp

Required. The RenderingApplication object.

bstrLinkPattern

Required. String. The pattern string used to generate the designated form strings.

varLanguageDirectory

Optional. String.

Remarks

The **ListComposeForms** method can be used to support launching custom forms from Web clients.

The *bstrLinkPattern* parameter can include any or all of the following substitution tokens:

| Substitution token | Attribute substituted for token |
|--------------------|-----------------------------------|
| %class% | The message class |
| %classpath% | The partial URL path for the form |
| %formname% | The form's name from FORM.INI |

Each returned string is a copy of the *bstrLinkPattern* string with any substitution tokens replaced with the values appropriate for the corresponding form.

For example, Microsoft® Outlook™ Web Access might set *bstrLinkPattern* to

```
rgForms = objRA.ListComposeForms("<A HREF='JavaScript:openNewWindow  
('"/Exchange/forms/%classpath%/frmroot.asp?command=new",  
"New message"', 640, 500)'>%formname%</A>")
```

In order for a form to be reflected in the returned list, there must be a file named FORM.INI in the form directory, as well as frmroot.asp. FORM.INI may have the following contents:

```
[Description]  
DisplayName=<form name>  
Hidden=0/1
```

If FORM.INI files have been added to or deleted from the form directory since you called **ListComposeForms**, you can refresh the list of designated compose forms by setting the **FormsRoot** property to its current value.

Example

This script demonstrates the **ListComposeForms** method as it might be used from Microsoft® Outlook™ Web Access:

```
<HTML>  
<!--
```

```
This script tests RenderingApplication.ListComposeForms  
First it writes out a table listing all the %tokens% for each form.
```

Next it writes out a table using an Outlook Web Access link pattern;
these links should bring up a form when followed.

```
-->
<HEAD>
<META NAME="GENERATOR" Content="Microsoft Developer Studio">
<META HTTP-EQUIV="Content-Type" content="text/html; charset=iso-8859-1">
<TITLE>Test RenderingApplication.ListComposeForms</TITLE>
<script language="Javascript">

function openNewWindow(fileName,windowName,theWidth,theHeight) {
  if (windowName == "newMessageWindow")
  {
    //generate random window ID
    windowName = new String(Math.round(Math.random() * 100000));
  }
  window.open(fileName,windowName,"toolbar=0,location=0,directories=0,
  status=1,menubar=1,scrollbars=1,resizable=1,width="+theWidth+",
  height="+theHeight)
}
</script>
</HEAD>
<BODY>
<%
set objRA = Application("RenderApplication")
szCommand = Request.QueryString("command")
%>
<% if szCommand = "refresh" then %>
<%
Err.Clear
objRA.FormsRoot = Server.MapPath("/Exchange") & "\usa"
if err <> 0 then
Response.Write err.number & " : " & err.description & "<br>"
else
Response.Write "Forms list successfully refreshed." & "<br>"
end if
%>
<center>
<P>Click <A HREF=f.asp>here</A> to view the forms list.
</center>
<% else %>
<%
rgNames = objRA.ListComposeForms("%formname%")
rgPaths = objRA.ListComposeForms("%classpath%")
rgClasses = objRA.ListComposeForms("%class%")
rgForms = objRA.ListComposeForms("<A HREF='JavaScript:openNewWindow
('"/Exchange/forms/%classpath%/frmroot.asp?command=new"',
""New message"", 640, 500)'">%formname%</A>")
%>
<% if isempty(rgnames) then %>
<P>
<% else %>
<P>This table enumerates all the variables accessible through ListComposeForms.
<table border=1 columns=3 width=100%>
<tr><td><b>Name</b></td><td><b>Path</b></td><td><b>Class</b></td></tr>
<%
```

```

for i = 1 to UBound(rgNames)
  Response.Write "<tr>"
  Response.Write "<td>" & rgNames(i) & "</td><td>" & rgPaths(i) &
    "</td><td>" & rgClasses(i) & "</td>"
  Response.Write "</tr>"
next
%>
</table>
<% end if %>
<% if isempty(rgForms) then %>
<P>There are no composable forms on this server.
<% else %>
<P>This table contains working links to forms.
<center>
<table border=1 width=50%>
<%
  for each szForm in rgForms
    Response.Write "<tr><td align=center>"
    Response.Write szForm
    Response.Write "</td></tr>"
  next
%>
</table>
</center>
<% end if %>
<center>
<P>Click <A HREF=f.asp?command=refresh>here</A> to refresh the forms list.
</center>
<% end if %>
</BODY>
</HTML>

```

LoadConfiguration Method (RenderingApplication Object)

Group

The **LoadConfiguration** method loads configuration information from the specified source.

Syntax

objRenderApp.**LoadConfiguration**(*eSource*, *bstrSection* [, *varSession*])

objRenderApp

Required. The RenderingApplication object.

eSource

Required. Integer. The enumerated value of the CDOHTML configuration source. The *eSource* parameter can have exactly one of the following values:

| eSource setting | Decimal value | Meaning |
|--------------------------|----------------------|---|
| CdoConfigRegistry | 1 | The standard registry key name for the CDO Rendering Library. |
| CdoConfigDS | 2 | The Microsoft® Exchange directory server. |

bstrSection

Required. String. The name of the section in the configuration source to load the information from. This parameter is ignored if the *eSource* parameter is set to **CdoConfigDS**.

varSession

Optional. Object. The Session object the messaging user is logged on to, if any. This parameter is ignored if the *eSource* parameter is set to **CdoConfigRegistry**.

Remarks

The **LoadConfiguration** method is normally called twice. A newly created rendering application calls **LoadConfiguration** with the *eSource* parameter set to **CdoConfigRegistry** to read data from the registry, including the location of the directory server. Later, when a session is started, **LoadConfiguration** is called with *eSource* set to **CdoConfigDS** to read information from the protocol settings for HTTP in the directory server's database.

If the *eSource* parameter is **CdoConfigRegistry**, the *bstrSection* parameter should be the key to the registry section containing values for Enterprise, Site, and Server. A standard value for this key name is

HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\MSExchangeWEB\Parameters

If the *eSource* parameter is set to **CdoConfigDS**, the *bstrSection* parameter is ignored because the directory server only has one configuration object. The registry information must already be loaded before you call **LoadConfiguration** with **CdoConfigDS**, since the Enterprise, Site, and Server must be set in order to read from the directory server.

When *eSource* is **CdoConfigDS**, the **LoadConfiguration** method normally does an anonymous Logon to obtain directory server information. If you are already logged on to a session, you can pass the Session object in the *varSession* parameter so that **LoadConfiguration** does not have to log on again. This can improve performance.

All values read in by **LoadConfiguration** can be retrieved with the ConfigParameter method.

LoggingLevel Property (RenderingApplication Object)

Group

The **LoggingLevel** property returns or sets the verbosity level for the specified logging category. Read/write.

Syntax

objRendApp.LoggingLevel(*category*)

category

Required. Long. The logging category.

Data Type

Long

Remarks

The **LoggingLevel** property controls how much information is written to the event log.

The logging level construct is an array with five elements, one for each logging category. Each element in the array can have a value from 0 to 5, representing the logging verbosity for that category.

The verbosity level bears an inverse relationship to the severity of the events being logged. The lower the verbosity level for a category, the fewer events are logged, that is, only the more severe ones. Each successive logging level logs all events logged by lower levels and also includes events of lesser severity introduced at its own level.

Level 0, the default, is the least verbose and logs only the most severe errors for that logging category. Level 5 is the most verbose and logs all events at all levels of severity.

The logging level for each category can have the following values:

Verbosity

| level | Event logging in this category |
|-------|---|
| 0 | Critical - log only the most severe failure events (default). |
| 1 | Minimal - include nearly all error events. |
| 2 | Basic - include certain important success events. |
| 3 | Extensive - include most routine success events. |
| 4 | Verbose - include all events not related to internal workings. |
| 5 | Internal - include events of interest only to users familiar with the internal workings of the CDO Rendering Library. |

The logging categories are as follows:

| Logging category | Value | Meaning |
|------------------|-------|--|
| CATEGORY_STARTUP | 1 | Events that occur during the creation of the rendering object |
| CATEGORY_GENERAL | 2 | Events that occur while the rendering object is generating HTML output |

Name Property (RenderingApplication Object)

The **Name** property returns or sets the display name of this rendering application. Read/write.

Syntax

objRenderApp.Name

Data Type

String

Remarks

The **Name** property originally contains the string "Microsoft CDO Rendering 1.2.1 Library". It should normally be left unchanged, but it can be modified to distinguish between entries in an event log.

Version Property (RenderingApplication Object) Group

The **Version** property returns a string representing the current version of the CDOHTML.DLL library. Read-only.

Syntax

objRenderApp.Version

Data Type

String

Remarks

The **Version** property contains the string "1.1" in the current release of the CDO Rendering Library.

VirtualRoot Property (RenderingApplication Object)

Group

The **VirtualRoot** property returns or sets the beginning of a URL. Read/write.

Syntax

objRendApp.VirtualRoot

Data Type

String

Remarks

The **VirtualRoot** property contains the common beginning of a set of HTTP paths, which normally correspond to disk paths on the server specified in the URL. **VirtualRoot** should always start with a forward slash (/).

Setting the **VirtualRoot** property causes the rendering application to create a special-purpose format with the name "virtroot". This format contains the string that was last set in **VirtualRoot**. The %virtroot % format can be used as a common beginning for HTTP paths when creating links to objects such as attachments.

Note that the **VirtualRoot** property contains an HTTP path at the browser, while the **FormsRoot** property contains an absolute disk directory path at Microsoft® Internet Information Server (IIS).

The **VirtualRoot** property defaults to "/exchange".

TableView Object

The TableView object represents a tabular view of an address book container or a folder.

At a Glance

| | |
|----------------------------|-----------------------------------|
| Specified in type library: | CDOHTML.DLL |
| First available in: | CDO Rendering Library version 1.1 |
| Parent objects: | <u>Views</u> collection |
| Child objects: | <u>Columns</u> collection |
| Default property: | (none) |

Properties

| Name | Available since version | Type | Access |
|-------------------|-------------------------|--|------------|
| <u>Categories</u> | 1.1 | Long | Read/write |
| <u>Class</u> | 1.1 | Long | Read-only |
| <u>Columns</u> | 1.1 | Column object or Columns collection object | Read-only |
| <u>Index</u> | 1.1 | Long | Read-only |
| <u>Name</u> | 1.1 | String | Read-only |
| <u>Parent</u> | 1.1 | Views collection object | Read-only |
| <u>Source</u> | 1.1 | Long | Read-only |

Methods

| Name | Available since version | Parameters |
|-----------------|-------------------------|---------------------------------|
| <u>IsSameAs</u> | 1.1 | <i>varView</i> as Object |

Remarks

A table view is a specification of a tabular rendering for a container object. The container object can be an address book container or a folder. The table view is applied to the container object in the context of a ContainerRenderer. The container renderer specifies the container object in its **DataSource** property and the table view to be applied in its **CurrentView** property.

The table view in turn contains a collection of Column objects. The collection is obtainable from the table view's **Columns** property. Each Column object specifies a property to be rendered in its **Property** property and the manner of rendering that property in its **RenderUsing** property. The display order of the columns is determined by the ordering of the table view's Columns collection. The leftmost column is the one obtained from the collection's **Item** property with an *index* value of 1, and the rightmost is obtained with an *index* value equal to the collection's **Count** property.

A table view is normally generated externally to a CDO application, although a nonpersistent table view

can be created with the **Add** method of the Views collection. A table view created in this way ceases to exist when the collection is released.

An externally generated table view can specify restrictions, sorting, and grouping. A restriction specifies which entries in the underlying container object are to be rendered. A sort specifies the order in which they are to be rendered. Grouping specifies how the sorted entries are to be categorized in the rendering.

A restriction is based on selected properties of the container object's entries and can be arbitrarily complex. A table view defined on a Messages collection, for example, can restrict the collection so that the only messages rendered are those that were received since February 6, 1998, have not yet been read, and have either a subject starting with "Bonus" or a message text containing "Bonus Calculation".

A table view's sort can be up to four levels deep. Each level corresponds to a property of the entries being sorted.

In addition to restrictions and sorting, a table view can be grouped, or categorized. Like a sort, a grouping can be up to four levels deep. Specifying grouping on a table view generates a sort on the same properties, in the same nesting order, as in the grouping.

If a restricted table view on a Messages collection is also grouped, GroupHeader objects are rendered along with the Message objects. Only the group headers corresponding to the messages that pass the restriction are rendered. AddressEntries collection views are not grouped, and only AddressEntry objects are rendered.

Categories Property (TableView Object)

Group

The **Categories** property returns or sets the number of categories in this table view. Read/write.

Syntax

objTableView.Categories

Data Type

Long

Remarks

The **Categories** property indicates how many levels of grouping are present in the table view. If the view is grouped, or categorized, **Categories** can contain from 1 through 4, the maximum permitted grouping depth. If the view is not categorized, **Categories** contains 0.

The nesting depth of a particular group within the table view is given by the **Level** property of the **GroupHeader** object that heads that group.

Columns Property (TableView Object)

Group

The **Columns** property returns a single Column object or a Columns collection for this table view. Read-only.

Syntax

Set *objColumns* = *objTableView*.**Columns**

Set *objColumn* = *objTableView*.**Columns**(*index*)

objColumns

Object. The Columns collection of this table view.

objTableView

Required. The TableView object.

objColumn

Object. An individual Column object belonging to this table view's Columns collection.

index

Integer. An index into the table view's Columns collection.

Data Type

Object (Column or Columns collection)

Remarks

If a Column object is to be accessed with the *index* parameter, the value of *index* must be between 1 and the size of the TableView object's Columns collection. This size is available in the collection's Count property.

Although the **Columns** property itself is read-only, the collection it returns can be accessed in the normal manner through its **Add** method, and the properties on its member Column objects retain their respective read/write or read-only accessibility.

Index Property (TableView Object)

Group

The **Index** property returns the index number for this TableView object within the Views collection. Read-only.

Syntax

objTableView.Index

Data Type

Long

Remarks

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

The first view in the Views collection has a **Index** value of 1.

An index value should not be considered a static value that remains constant for the duration of a container renderer. It can be affected when other views are added and deleted.

IsSameAs Method (TableView Object)

Group

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

Syntax

boolSame = *objTableView*.**IsSameAs**(*varView*)

boolSame

On successful return, contains **True** if the two objects are the same.

objTableView

Required. This TableView object.

varView

Required. Object. The view object being compared against.

Remarks

The *varView* parameter should be declared as an **Object** rather than as a **TableView**. This allows for comparison among different classes of view objects being held in a Views collection.

Two view objects are considered to be the same if and only if their pointer values are the same, that is, if and only if they are the identical object. Otherwise **IsSameAs** returns **False**.

Name Property (TableView Object)

Group

The **Name** property returns the display name of this TableView object. Read-only.

Syntax

objTableView.Name

Data Type

String

Remarks

The **Name** property represents the display name assigned to this table view. It can be used to refer to the table view, and to retrieve it by name using the container renderer's **CurrentView** property or the Views collection's **Item** property.

You should give every view object a unique name. CDO does not enforce uniqueness, but if a collection has duplicate names, only the first one can be found by name.

Source Property (TableView Object)



The **Source** property returns the type of this table view. Read-only.

Syntax

objTableView.Source

Data Type

Long

Remarks

The **Source** property indicates the source of the definition of the table view. It can have exactly one of the following values:

| Table view source | Decimal value | Meaning |
|------------------------|---------------|---|
| CdoViewCommon | 0 | This table view is predefined globally for all folders and all messaging users. |
| CdoViewCustom | 2 | This table view has been defined in the context of the current setting of the DataSource property of the <u>ContainerRenderer</u> object. However, it is still available when the DataSource property is changed. |
| CdoViewFolder | 3 | This table view is predefined for the particular folder currently being rendered. It is no longer available when the DataSource property is changed. |
| CdoViewPersonal | 1 | This table view is predefined for the messaging user associated with the current session represented by the <u>Session</u> object. |

For more information on table view rendering, see Rendering Container Objects.

Views Collection Object

The Views collection object contains one or more views for a container object.

At a Glance

| | |
|----------------------------|---|
| Specified in type library: | CDOHTML.DLL |
| First available in: | CDO Rendering Library version 1.1 |
| Parent objects: | ContainerRenderer |
| Child objects: | CalendarView
TableView |
| Default property: | Item |

An Views collection supports count and index values that let you access an individual TableView object through the **Item** property. The Views collection also supports the Microsoft® Visual Basic® **For Each** statement.

Properties

| Name | Available since version | Type | Access |
|------------------------|-------------------------|--------------------------|-----------|
| Class | 1.1 | Long | Read-only |
| Count | 1.1 | Long | Read-only |
| Item | 1.1 | TableView object | Read-only |
| Parent | 1.1 | ContainerRenderer object | Read-only |

Methods

| Name | Available since version | Parameters |
|---------------------|-------------------------|--|
| Add | 1.1 | <i>bstrName</i> as String ,
(optional) <i>varClass</i> as Long ,
(optional) <i>varSortBy</i> as Variant ,
(optional) <i>varSortAscending</i> as Boolean |

Remarks

The Views collection can contain a variety of different classes of view objects. The classes currently implemented are represented by the [CalendarView](#) and [TableView](#) objects.

The Views collection is used by a [ContainerRenderer](#) object to render a container object, such as an address book container or a folder. The Views collection comes into being when a rendering application sets the container renderer's **DataSource** property to the container object. The collection is released when the parent ContainerRenderer object is released, or when a new container object is set in the **DataSource** property.

The classes of views that can be held in a Views collection and rendered by a ContainerRenderer object are as follows:

View classCalendarViewTableView**View source container object**Messages collection containing AppointmentItem objectsAddressEntries collection, Folders collection, Messages collection, or Recipients collection

The various view objects initially in the collection are those that were already generated externally to the rendering application. These views persist in the underlying store, typically a directory or message store. New views can be defined and contributed to the collection using the **Add** method, but they do not persist after the collection is released.

The view to be applied to the container object is specified in the container renderer's **CurrentView** property. A newly instantiated Views collection always has a default current view. This default can be specified by the store underlying the container object. If it is not specified, the CDO Rendering Library sets it to the first view in the collection.

Changing the current view causes a new AddressEntries, Folders, Messages, or Recipients collection to be instantiated. This collection contains only the items that pass the restriction specified by the new view. The AddressEntry, Folder, Message, or Recipient objects in the collection are sorted as specified by the view. If a folder view is categorized, GroupHeader objects appear in the collection along with the messages.

The initial filter on the AddressEntries, Folders, Messages, or Recipients collection is inherited from the view's restriction. It can be used without modification, but it cannot be read or changed by the rendering application. Any attempt to read a property on an inherited AddressEntryFilter or MessageFilter object results in an error return. Writing any property on an inherited filter disinherits it and refreshes the collection. This means that the collection is reinstantiated with a new filter specifying only the property just written. This new filter, however, is no longer inherited, and the application can read its properties and set additional restrictions within it.

Add Method (Views Collection)

Group

The **Add** method creates and returns a new CalendarView or TableView object in the Views collection.

Syntax

```
Set objView = objViewsColl.Add(bstrName [, varClass] [, varSortBy] [, varSortAscending] )
```

objView

On successful return, contains the new CalendarView or TableView object.

objViewsColl

Required. The Views collection object.

bstrName

Required. String. The display name to be assigned to the new CalendarView or TableView object.

varClass

Optional. Long. The class of the view to be created. The currently supported classes are **CdoClassCalendarView** and **CdoClassTableView**. The default value is **CdoClassTableView**.

varSortBy

Optional. Variant (Long or String). The property on which to sort the new view. The default value is **CdoPR_MESSAGE_DELIVERY_TIME** if a corresponding column is present in the view. If no such column is present and the *varSortBy* parameter is not furnished, there is no default value and the sort is undefined.

varSortAscending

Optional. Boolean. The sort direction of the new view. Set to **True** to sort in ascending order. The default value is **False**.

Remarks

The **Add** method's *bstrName* parameter corresponds to the **Name** property of the new CalendarView or the **Name** property of the new TableView object. The *varClass* parameter corresponds to the new view's **Class** property.

The calendar view's **Name** property and the table view's **Name** property are both read-only. After you set the display name of a new view, you cannot subsequently change it.

You should give every view object a unique name. CDO does not enforce uniqueness, but if a collection has duplicate names, only the first one can be found by name.

The *varSortBy* parameter must designate a property represented by a Column object within the new view's Columns collection. *varSortBy* must contain the property tag of the desired property. If you wish to sort the view on a custom property, you can obtain its property tag from the **ID** property of the Field object being used to access the custom property:

An address book container does not have any common, personal, or folder views predefined. If you are rendering an address book container, you must define one or more custom views for it. Furthermore, you must **Add** any new custom views to the present Views collection before you set the ContainerRenderer object's **DataSource** property to an AddressEntries collection. Otherwise the new custom views are not handled properly in the rendering.

Example

This code fragment creates a custom table view to sort the messages in the Inbox folder according to their index values in the Inbox Messages collection. A new field must be created and set to each message's index value, because Message objects do not expose an **Index** property. Then the **ID** property of the new field is used to specify the sort property for the new view. Finally, a corresponding column is added to the view's Columns collection using its **Add** method. This is necessary because the sort property must be represented by a Column object in the view. The new column cannot be added

until after the new view has been created, but the column criterion for the *varSortBy* parameter is considered satisfied as long as the column is added before the view is rendered.

```
' assume Session and ContainerRenderer already valid
Dim colMsgs As Messages ' Inbox collection
Dim objMsg As Message
Dim colFields As Fields ' Message object fields
Dim objMsgIx As Field ' custom field for message index
Dim index As Long ' index of each message in Inbox collection
Dim tagMsgIx As Long ' property tag of custom message index field
Dim objIxView As TableView ' new table view sorted on index value

Set colMsgs = objSess.Inbox.Messages
objContRend.DataSource = colMsgs ' render Inbox Messages collection
index = 0
For Each objMsg in colMsgs
index = index + 1
    Set colFields = objMsg.Fields
    ' data type vbLong has decimal value 3:
    Set objMsgIx = colFields.Add ("Message index", 3, index)
    objMsg.Update ' save new field in this message
Next
tagMsgIx = colFields.Item(objMsgIx.Index).ID
' class CdoTableview has decimal value 9:
Set objIxView = objContRend.Views.Add _
    "Table view on message index", 9, tagMsgIx
objIxView.Columns.Add "Index", tagMsgIx, 3, 0, 0, 3
```

Count Property (Views Collection)

Group

The **Count** property returns the number of [CalendarView](#) or [TableView](#) objects in the collection. Read-only.

Syntax

objViewsColl.Count

Data Type

Long

Remarks

For more information on using the **Count** and **Item** properties, see the example in the [Item](#) property.

Item Property (Views Collection)

Group

The **Item** property returns the specified [CalendarView](#) or [TableView](#) object from the Views collection. Read-only.

Syntax

objViewsColl.**Item**(*index*)

objViewsColl.**Item**(*name*)

index

A short integer ranging from 1 to *objViewsColl*.**Count**.

name

The display name of the [CalendarView](#) or [TableView](#) object to be selected from the Views collection.

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

Data Type

[TableView](#) object

Remarks

The **Item** property works like an accessor property.

If the specified view object is not found in the collection, the **Item** property returns **Nothing**.

The names of the predefined calendar views are "Daily" for a view with mode **CdoModeCalendarDaily** and "Weekly" for a view with mode **CdoModeCalendarWeekly**. The mode of the view is available from its **Mode** property.

You should give every view object a unique name. CDO does not enforce uniqueness, but if a collection has duplicate names, only the first one can be found by name.

Although the **Item** property itself is read-only, the [CalendarView](#) or [TableView](#) object it returns can be accessed in the normal manner, and its properties retain their respective read/write or read-only accessibility.

Example

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

```
' Put all view names in a collection into a string array
Dim strItemName(100) As String
Dim i As Integer ' loop counter
' error handling omitted from this fragment ...
For i = 1 To objViewsColl.Count Step 1
    strItemName(i) = objViewsColl.Item(i).Name
    ' or = objViewsColl(i) since Item and Name are default properties
    If 100 = i Then ' max size of string array
        Exit Function
    End If
Next i
```


CDO for NTS Library

The Microsoft® CDO for NTS Library (Collaboration Data Objects for Windows NT® Server) version 1.2.1 exposes messaging objects for use by Microsoft® Visual Basic®, C/C++, Microsoft® Visual C++®, and Visual Basic Scripting Edition (VBScript) applications. The library allows server applications to send and receive messages without requiring access to the Microsoft® Exchange Server. You can create programmable messaging objects, then use their properties and methods for sending and receiving.

The CDO for NTS Library is intended to run on a Microsoft® Windows NT® Server, for example from Active Server Pages (ASP) script on a Microsoft® Internet Information Server (IIS). It is not intended to run on a client process, nor to access remote servers. No user dialog is invoked or supported by CDO for NTS.

The CDO for NTS Library interfaces with the SMTP (Simple Mail Transfer Protocol) server component of Microsoft® Internet Information Server (IIS) version 4.0 and later. The Session object uses the **LogonSMTP** method to differentiate the access from the **Logon** method of the the CDO Library, which interfaces with Microsoft® Exchange Server.

The SMTP server component of IIS has its own message store mechanism. The Inbox and Outbox are mapped to directories in the file system, and no other folders exist. Message transfer takes place in such a way that spooling appears instantaneous, so the Inbox has no incoming queue and the Outbox is always empty.

When CDO for NTS is running with IIS, the Inbox is a single common folder shared by all SMTP recipients and applications. It contains all messages received by IIS and destined for the local domains the SMTP server is configured for. However, the incoming messages are segregated by the CDO for NTS Library according to their recipients. An application can only access messages destined for the address it used when it logged on.

When CDO for NTS is running with the Microsoft Exchange Server, the Inbox is the regular Inbox of the messaging user's mailbox. When CDO for NTS is running with Microsoft MCIS 2.0 Mail, the Inbox is the messaging user's Post Office Protocol version 3 (POP3) server Inbox.

Applications developed to run with CDO for NTS can also run with CDO for Exchange provided they do not use the NewMail object. Also, the Session object's **LogonSMTP** method should be changed to the CDO for Exchange session's **Logon** method.

Comparison of CDO Libraries

The CDO Library and the CDO Rendering Library are often used in conjunction with each other to accomplish a wide variety of tasks. The CDO for NTS Library is intended as a streamlined alternative for a frequently used subset of these tasks. It operates independently of the other two libraries.

All three CDO libraries have features useful to applications in different circumstances. The following table compares the features of the CDO for NTS Library with those of the CDO Library used with the CDO Rendering Library:

| Feature | CDO for NTS Library | CDO and CDO Rendering Libraries |
|---|----------------------------|--|
| Based on MAPI | No | YES |
| Supports profiles | No | YES |
| Supports authenticated users | No | YES |
| Supports address book access | No | YES |
| Supports remote server access | No | YES |
| Supports SMTP access | YES | No |
| Supports calendar access | No | YES |
| Supports HTML rendering | No | YES |
| Supports MHTML messages | YES | No |
| Supports NewMail object | YES | No |
| Can be called from ASP | YES | YES |
| Compatible with IIS 4.0 and MCIS | YES | No |
| Compatible with Active Messaging 1.1 applications | (partially) | YES |

The CDO for NTS Library is functionally compatible with a subset of the CDO Library version 1.2.1, and with a subset of the Active Messaging Library version 1.1. If an application has been written for either of these libraries, some modifications are usually required to run it with CDO for NTS. Some features of Active Messaging, such as fields, folders, and address lists, are unavailable in CDO for NTS.

A developer may be uncertain whether to use CDO for NTS or the CDO and CDO Rendering libraries. You should use CDO for NTS if you

- require support for MHTML;
- anticipate heavy automated generation of outgoing e-mail;
- intend to send unauthenticated e-mail from Web sites;
- use server-based replication of messages;
- anticipate using e-mail for server-based notifications.

You should use the CDO Library in conjunction with the CDO Rendering Library if you

- require authenticated user support;
- need to render objects or data into HTML;
- need to access a personal calendar store;
- intend to access multiple remote Microsoft® Exchange Servers;
- anticipate accessing e-mail from remote sites;
- use server-based custom mail agents;
- require automatic load balancing of outgoing mail;

- are primarily upgrading an existing CDO 1.2.1 or Active Messaging 1.1 application;
- are developing a client application;
- need to filter or sort messages;
- require support for multiple address book and message store providers;
- are developing a three-tier Web-based e-mail application.

You can use either CDO for NTS or the CDO and CDO Rendering libraries if you

- are developing a server application;
- intend to use server-based custom e-mail agents.

CDO for NTS Object Usage

The CDO for NTS objects are described in the remainder of this section. The following table summarizes the objects in alphabetic order and gives the principal function of each one.

| Object | Purpose |
|-------------------------------|---|
| <u>AddressEntry</u> | Specify addressing information for an individual messaging user. |
| <u>Attachment</u> | Associate an additional object with a message. |
| <u>Attachments</u> collection | Access all attachments on a message; create new attachments. |
| <u>Folder</u> | Open the default Inbox or Outbox folder in a message store. |
| <u>Message</u> | Compose, populate, send, and receive an e-mail document. |
| <u>Messages</u> collection | Access all messages in a folder; create new messages. |
| <u>NewMail</u> | Send a message without having to log on to a session. |
| <u>Recipient</u> | Specify information for a messaging user intended to receive a message. |
| <u>Recipients</u> collection | Access all recipients of a message; create new recipients. |
| <u>Session</u> | Establish a connection between an application and a messaging system. |

All these objects can be declared using **Object** for their data type, but it is preferable to take advantage of early binding, to enforce type checking and generate more efficient code:

```
Dim objMsg As Message ' in preference to Dim As Object
Dim colAtts As Attachments
```

You can also use an object's full type name in order to distinguish it from identically named objects available to a Visual Basic program through other object libraries such as the CDO Library:

```
Dim objSess As CDONTS.Session ' distinguished from MAPI.Session
Dim colRecips As CDONTS.Recipients
```

Installing CDO for NTS

To use the CDO for NTS Library, make sure it is referenced by the automation controller you are using. With Microsoft Visual Basic, for example, run the **References** command and select the check box for **Microsoft CDO for NTS 1.2.1 Library**.

To successfully run the CDONTS.DLL file, which holds both the in-process server and the type library, you must have installed Microsoft® Internet Explorer version 4.0 or later. A minimal installation is sufficient, and the Internet Explorer does not have to be actually running.

CDO for NTS is automatically installed during the typical installation of IIS 4.0. Microsoft Exchange does not have to be installed at the IIS computer on which CDO for NTS is being installed.

To run with Microsoft Exchange version 5.5, a minimal installation of Internet Explorer 4.0 is still required on the server. If the Microsoft Exchange Server is already installed, you can install Internet Explorer 4.0 on the server and then run the Internet Mail Service (IMS) Setup Wizard to enable CDO for NTS. This configures CDO for NTS to use the SMTP services that Exchange provides.

If you are installing the Microsoft Exchange Server on the same machine as IIS 4.0, Internet Explorer 4.0 should already be installed. You must also run the IMS Wizard in order to configure CDO for NTS correctly. If the IMS Wizard is not run, CDO for NTS continues to use the SMTP services provided by IIS 4.0.

For proper delivery of mail, you must have the SMTP server component of IIS configured to send mail to the addresses you specify. This is done through the Domain Name System (DNS). For more information, see *Microsoft SMTP Service* in the *Microsoft Windows NT 4.0 Option Pack Documentation*.

CDO for NTS Objects, Properties, and Methods

This reference contains property and method information for the CDO for NTS Library objects.

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

| Object | Available since version | Properties | Methods |
|-------------------------------|--------------------------------|---|--|
| <u>AddressEntry</u> | 1.2 | Address, Application, Class, Name, Parent, Session, Type | (none) |
| <u>Attachment</u> | 1.2 | Application, Class, ContentBase, ContentID, ContentLocation, Name, Parent, Session, Source, Type | Delete, ReadFromFile, WriteToFile |
| <u>Attachments collection</u> | 1.2 | Application, Class, Count, Item, Parent, Session | Add, Delete |
| <u>Folder</u> | 1.2 | Application, Class, Messages, Name, Parent, Session | (none) |
| <u>Message</u> | 1.2 | Application, Attachments, Class, ContentBase, ContentID, ContentLocation, HTMLText, Importance, MessageFormat, Parent, Recipients, Sender, Session, Size, Subject, Text, TimeReceived, TimeSent | Delete, Send |
| <u>Messages collection</u> | 1.2 | Application, Class, Count, Item, Parent, Session | Add, Delete, GetFirst, GetLast, GetNext, GetPrevious |
| <u>NewMail</u> | 1.2 | Bcc, Body, BodyFormat, Cc, ContentBase, ContentLocation, From, Importance, MailFormat, | AttachFile, AttachURL, Send, SetLocaleIDs |

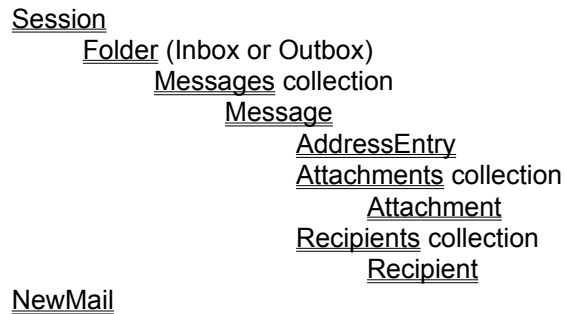
| | | | |
|----------------------------------|-----|---|--|
| | | Subject, To, Value,
Version | |
| <u>Recipient</u> | 1.2 | Address,
Application, Class,
Name, Parent,
Session, Type | Delete |
| <u>Recipients
collection</u> | 1.2 | Application, Class,
Count, Item,
Parent, Session | Add, Delete |
| <u>Session</u> | 1.2 | Application, Class,
Inbox,
MessageFormat,
Name, Outbox,
Parent, Session,
Version | GetDefaultFolder,
Logoff,
LogonSMTP,
SetLocaleIDs |

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.

CDO for NTS Object Model

The object model for the CDO for NTS Library is hierarchical. The following table shows the containment hierarchy. Each indented object is 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.



The AddressEntry object is not hierarchically contained by the Message object, but its only access is through the message's **Sender** property, so it returns the Message object in its **Parent** property. The NewMail object is independent of the rest of the hierarchy and does not access any of the other objects.

Properties Common to All CDO for NTS Library Objects

All Microsoft® CDO for NTS Library objects except the NewMail object expose 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 CDO for NTS Library object.

The NewMail object is self-contained and does not expose any of these properties.

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 CDO for NTS 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 CDO for NTS Library objects and that have the same meaning for all objects.

Properties

| Name | Type | Access |
|--------------------|----------------|-----------|
| <u>Application</u> | String | Read-only |
| <u>Class</u> | Long | Read-only |
| <u>Parent</u> | Object | Read-only |
| <u>Session</u> | Session object | Read-only |

Application Property (All CDONTS Library Objects) Group

The **Application** property returns the name of the active application, namely the Microsoft® CDO for NTS Library. Read-only.

Syntax

object.**Application**

Data Type

String

Remarks

The **Application** property always contains the string "Collaboration Data Objects for NTS version 1.2.1".

The version number of the CDO for NTS 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 ' or Dim objSession As CDONTS.Session
' error handling ...
Set objSession = CreateObject("CDONTS.Session")
If Not objSession Is Nothing Then
    objSession.LogonSMTP("My Name", "myaddress@mycompany.com")
    MsgBox "Session's Application property = " & objSession.Application
End If
' error handling
End Function
```

Class Property (All CDONTS Library Objects)

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 CDO for NTS Library object. The following values are defined:

| CDO for NTS Library object | Class value | Type library constant |
|-------------------------------|-------------|------------------------|
| <u>AddressEntry</u> | 8 | CdoAddressEntry |
| <u>Attachment</u> | 5 | CdoAttachment |
| <u>Attachments</u> collection | 18 | CdoAttachments |
| <u>Folder</u> | 2 | CdoFolder |
| <u>Message</u> | 3 | CdoMsg |
| <u>Messages</u> collection | 16 | CdoMessages |
| <u>Recipient</u> | 4 | CdoRecipient |
| <u>Recipients</u> collection | 17 | CdoRecipients |
| <u>Session</u> | 0 | CdoSession |

Example

```
' Function: Util_DecomposeObjectClass
' Purpose: Decompose the long integer class value,
'           show the related object name
' See documentation topic: Class property
Function Util_DecomposeObjectClass(IClass As Long)
' error handling here ...
Select Case (IClass)
    Case CdoSession:
        MsgBox ("Session object; Class = " & IClass)
    Case CdoMsg:
        MsgBox ("Message object; Class = " & IClass)
End Select
' error handling here ...
End Function
```

```
' Function: TestDrv_Util_DecomposeObjectClass
' Purpose: Call the utility function DecomposeObjectClass for Class values
' See documentation topic: Class property
Function TestDrv_Util_DecomposeObjectClass()
' error handling here ...
If objSession Is Nothing Then
```

```
    MsgBox "Need to set the Session object: Session->LogonSMTP"  
    Exit Function  
End If  
' expect type CdoSession = 0 for Session object  
Util_DecomposeObjectClass (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 CdoMessage = 3 for Message object  
Util_DecomposeObjectClass (objOneMsg.Class)  
' error handling here ...  
End Function
```

Parent Property (All CDONTS Library Objects)

Group

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 CDO for NTS Library returns the *immediate* parent of an object. The immediate parent for each object is shown in the following table.

| CDO for NTS Library object | Immediate parent in object hierarchy |
|--|--|
| AddressEntry | Message |
| Attachment | Attachments collection |
| Attachments collection | Message |
| Folder | Session |
| Message | Messages collection |
| Messages collection | Folder , including 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 CDO for NTS Library objects and its **Parent** property is set to **Nothing**. The [AddressEntry](#) object does not have a true hierarchical parent but can only be obtained through the **Sender** property of a [Message](#) object. Its **Parent** property returns the Message object.

For more information on the CDO for NTS Library object hierarchy, see [CDO for NTS 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 Messages collection available"

 Exit Function

End If

MsgBox "Messages collection parent class = " & objMessages.Parent.Class

Exit Function

' error handling here ...

End Function

Session Property (All CDONTS Library Objects)

Group

The **Session** property returns the top-level Session object associated with the specified CDO for NTS Library object. Read-only.

Syntax

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

Data Type

Object (Session)

Remarks

The Session object represents the highest level in the CDO for NTS Library object hierarchy. If you invoke the **Session** property of a Session object, it returns the same Session object.

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 folder supplied; 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
```

AddressEntry Object (CDONTS Library)

The AddressEntry object defines addressing information valid for a given messaging system.

At a Glance

| | |
|----------------------------|---|
| Specified in type library: | CDONTS.DLL |
| First available in: | CDO for NTS Library version 1.2 |
| Parent objects: | Message (AddressEntry object only obtainable through Sender property of Message object) |
| Child objects: | (none) |
| Default property: | Name |

Properties

| Name | Available since version | Type | Access |
|-----------------------------|-------------------------|----------------|-----------|
| Address | 1.2 | String | Read-only |
| Application | 1.2 | String | Read-only |
| Class | 1.2 | Long | Read-only |
| Name | 1.2 | String | Read-only |
| Parent | 1.2 | Message object | Read-only |
| Session | 1.2 | Session object | Read-only |
| Type | 1.2 | String | Read-only |

Methods

(None.)

Remarks

An address usually represents a person or process to which the messaging system can deliver messages.

The AddressEntry object is only available through the [Sender](#) property of a [Message](#) object. It is used to obtain the sender's name, e-mail address, and address type for use in constructing a new [Recipient](#) object on an outbound message.

Address Property (CDONTS AddressEntry Object) Group

The **Address** property specifies the messaging address of an address entry. Read-only.

Syntax

objAddressEntry.**Address**

Data Type

String

Remarks

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

Example

```
' Set up a series of object variables
' assume valid Session object
Set objInbox = objSession.GetDefaultFolder(CdoDefaultFolderInbox)
Set collInMessages = objInbox.Messages
Set objMessage = collInMessages.GetFirst
Set objAddrEntry = objMessage.Sender
strMsg = "Sender name " & objAddrEntry.Name
strMsg = strMsg & "; address type = " & objAddrEntry.Type
strMsg = strMsg & "; e-mail address = " & objAddrEntry.Address
MsgBox strMsg
```

Name Property (CDONTS AddressEntry Object)

Group

The **Name** property returns the display name or alias of the AddressEntry object as a string. Read-only.

Syntax

objAddressEntry.**Name**

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

Data Type

String

Example

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

Type Property (CDONTS AddressEntry Object)

Group

The **Type** property specifies the address type, such as SMTP. Read-only.

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.

The **Type** property always returns SMTP in the current version of CDO for NTS.

Example

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

Attachment Object (CDONTS Library)

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

At a Glance

| | |
|----------------------------|---------------------------------|
| Specified in type library: | CDONTS.DLL |
| First available in: | CDO for NTS Library version 1.2 |
| Parent objects: | <u>Attachments</u> collection |
| Child objects: | (none) |
| Default property: | <u>Name</u> |

No modifications are permitted to messages in the Inbox, except for message deletion. Therefore, all properties on an Attachment object are treated as read-only when the grandparent Message object is in the Inbox. When the access is described as read/write in the following table, that access applies only to attachments on messages in the Outbox.

Properties

| Name | Available since version | Type | Access |
|------------------------|-------------------------|-------------------------------|------------|
| <u>Application</u> | 1.2 | String | Read-only |
| <u>Class</u> | 1.2 | Long | Read-only |
| <u>ContentBase</u> | 1.2 | String | Read-only |
| <u>ContentID</u> | 1.2 | String | Read-only |
| <u>ContentLocation</u> | 1.2 | String | Read-only |
| <u>Name</u> | 1.2 | String | Read/write |
| <u>Parent</u> | 1.2 | Attachments collection object | Read-only |
| <u>Session</u> | 1.2 | Session object | Read-only |
| <u>Source</u> | 1.2 | String or Message object | Read/write |
| <u>Type</u> | 1.2 | Long | Read/write |

Methods

| Name | Available since version | Parameters |
|---------------------|-------------------------|----------------------------------|
| <u>Delete</u> | 1.2 | (none) |
| <u>ReadFromFile</u> | 1.2 | <i>fileName</i> as String |
| <u>WriteToFile</u> | 1.2 | <i>fileName</i> as String |

Remarks

An attachment is an object, such as a file or another object, that is associated with and transmitted with a Message object. The Attachment object does not specify its location within the message. The client application makes all the display decisions for a message, including whether its attachments are to be displayed, and if so, when and where.

The CDO for NTS Library does not manage the actual display of the attachment. The properties of the Attachment object simply provide information that the displaying application can use to find and open the attachment, and to convert its contents into a display.

ContentBase Property (CDONTS Attachment Object) Group

The **ContentBase** property returns the Content-Base header of a MIME (Multipurpose Internet Mail Extensions) message attachment. Read-only.

Syntax

objAttach.**ContentBase**

Data Type

String

Remarks

The **ContentBase** property is used for MHTML (MIME HTML) support. It represents the Content-Base header for the appropriate MIME body part.

For more information on MHTML, see the RFC 2110 document.

ContentID Property (CDONTS Attachment Object) Group

The **ContentID** property returns the Content-ID header of a MIME (Multipurpose Internet Mail Extensions) message attachment. Read-only.

Syntax

objAttach.**ContentID**

Data Type

String

Remarks

The **ContentID** property is used for MHTML (MIME HTML) support. It represents the Content-ID header for the appropriate MIME body part.

The CDO for NTS Library does not generate Content-ID headers for outgoing messages or attachments. This property is provided to inspect an incoming message for Content-ID headers.

For more information on MHTML, see the RFC 2110 document.

ContentLocation Property (CDONTS Attachment Object) Group

The **ContentLocation** property returns the Content-Location header of a MIME (Multipurpose Internet Mail Extensions) message attachment. Read-only.

Syntax

objAttach.**ContentLocation**

Data Type

String

Remarks

The **ContentLocation** property is used for MHTML (MIME HTML) support. It represents the Content-Location header for the appropriate MIME body part.

For more information on MHTML, see the RFC 2110 document.

Delete Method (CDONTS Attachment Object)

Group

The **Delete** method removes the Attachment object from the Attachments collection.

Syntax

objAttach.Delete()

Remarks

The **Delete** method performs an irreversible operation on the collection. It calls **Release** on the collection's reference to the Attachment object. If you have another reference to the attachment, you can still access its properties and methods, but you can never again associate it with any collection because the **Add** method always creates a new object. You should **Set** your reference variable either to **Nothing** or to another attachment.

The final **Release** on the Attachment object takes place when you call **Delete** if you had no other reference, or when you assign your reference variable to **Nothing**. At this point the object is removed from memory. Attempted access to a released object results in an error return of **CdoE_INVALID_OBJECT**.

The effect of the **Delete** operation is not permanent until you use the **Send** or **Delete** method on the Message object to which this attachment belongs.

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.

The CDO for NTS Library does not permit any modifications to messages in the Inbox, other than deleting the entire message. Prohibited modifications include adding, deleting, or modifying any attachment; adding, deleting, or modifying any recipient; and modifying any message property, even one with read/write access.

Example

This code fragment illustrates the two situations previously explained. The **Set** statement calls **AddRef** on the first Attachment object. That reference survives the call to **Delete** and has to be reassigned. The second Attachment object is deleted without creating another reference, and no other action is necessary.

```
' assume valid Message object
Set objAttach = objMessage.Attachments.Item(1)
objAttach.Delete ' still have a reference from Set statement
' ... other operations on objAttachment possible but pointless ...
Set objAttach = Nothing ' necessary to remove reference
' ...
objMessage.Attachments.Item(2).Delete ' no reference to remove
```

Name Property (CDONTS Attachment Object)

Group

The **Name** property returns or sets the display name of the Attachment object as a string. Read/write (read-only for an attachment on a message in the Inbox).

Syntax

objAttach.Name

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

Data Type

String

Remarks

Before setting or changing the **Name** property, you should be sure that the **Source** property is already set. Setting **Source** after setting **Name** can result in an incorrect value for **Name**.

If the attachment's **Type** property is set to **CdoEmbeddedMessage**, any setting of the **Name** property is ignored in the current version of CDO for NTS. If the **MessageFormat** property of the grandparent **Message** object is set to **CdoMime**, the embedded message's **Subject** property is used for the attachment name. If the grandparent message's **MessageFormat** is **CdoText**, CDO for NTS randomly creates an attachment name in the format ATT00099.EML.

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

ReadFromFile Method (CDONTS Attachment Object) Group

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

Syntax

objAttach.**ReadFromFile**(*fileName*)

objAttach

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 |
|---------------------------|--|
| CdoFileData | Copies the contents of the specified file to the attachment. |
| CdoEmbeddedMessage | (Not supported) |

Note The current version of the CDO for NTS Library does not support **ReadFromFile** for **CdoEmbeddedMessage** attachments. These calls generate the run-time error **CdoE_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** method of the Attachments collection.

Source Property (CDONTS Attachment Object)

Group

The **Source** property returns or sets information specifying the location of the data for the attachment. Read/write (read-only for an attachment on a message in the Inbox).

Syntax

objAttach.**Source**

Data Type

String or Object (Message)

Remarks

The **Source** property is not used for **CdoFileData** attachments. For **CdoEmbeddedMessage** attachments, the **Source** property returns or sets the Message object to be embedded. An embedded message is copied into the attachment at creation time.

Note The **Source** property is a string except when it returns the source of a **CdoEmbeddedMessage** attachment.

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

| Type property | Source property |
|---------------------------|--|
| CdoFileData | Not used; contains an empty string. The source for this type of attachment must be specified in the call to the Add method. |
| CdoEmbeddedMessage | Specifies the <u>Message</u> object to be embedded. |

Note You must set Type before you set **Source**. Failure to do this can result in a return of **CdoE_NOT_FOUND** from the ReadFromFile or WriteToFile method.

The **Source** property can also be set at the time of creation of the attachment by supplying the *source* parameter to the **Add** method of the Attachments collection. For attachments of type **CdoFileData**, the **Add** method is the only place the source file can be specified. However, you can reset it later with the Attachment object's ReadFromFile method.

Type Property (CDONTS Attachment Object)

Group

The **Type** property describes the attachment type. Read/write (read-only for an attachment on a message in the Inbox).

Syntax

objAttach.Type

Data Type

Long

Remarks

The following attachment types are supported:

| Type property | Value | Description |
|---------------------------|-------|---|
| CdoFileData | 1 | Attachment is the contents of a file.
(Default value.) |
| CdoEmbeddedMessage | 4 | Attachment is an embedded message. |

The value of the **Type** property determines the valid values for the **Source** property. Consequently, you must set **Type** before setting **Source** in order for the **ReadFromFile** and **WriteToFile** methods to work correctly.

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

Example

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

WriteToFile Method (CDONTS Attachment Object) Group

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

Syntax

objAttach.**WriteToFile**(*fileName*)

objAttach

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 |
|---------------------------|--|
| CdoFileData | Copies the contents of the attachment to the specified file. |
| CdoEmbeddedMessage | (Not supported) |

Note The current version of the CDO for NTS Library does not support **WriteToFile** for **CdoEmbeddedMessage** attachments. These calls generate the run-time error **CdoE_NO_SUPPORT**.

Attachments Collection Object (CDONTS Library)

The Attachments collection object contains zero or more Attachment objects.

At a Glance

| | |
|----------------------------|---------------------------------|
| Specified in type library: | CDONTS.DLL |
| First available in: | CDO for NTS Library version 1.2 |
| Parent objects: | Message |
| Child objects: | Attachment |
| Default property: | Item |

Properties

| Name | Available since version | Type | Access |
|-----------------------------|-------------------------|-------------------|-----------|
| Application | 1.2 | String | Read-only |
| Class | 1.2 | Long | Read-only |
| Count | 1.2 | Long | Read-only |
| Item | 1.2 | Attachment object | Read-only |
| Parent | 1.2 | Message object | Read-only |
| Session | 1.2 | Session object | Read-only |

Methods

| Name | Available since version | Parameters |
|------------------------|-------------------------|--|
| Add | 1.2 | (optional) <i>name</i> as String ,
(optional) <i>type</i> as Long ,
(optional) <i>source</i> as String or Object ,
(optional) <i>ContentLocation</i> as String ,
(optional) <i>ContentBase</i> as String |
| Delete | 1.2 | (none) |

Add Method (CDONTS Attachments Collection)

Group

The **Add** method creates and returns a new [Attachment](#) object in the Attachments collection.

Syntax

Set *objAttach* = *collAttachments*.**Add**([*name*] [, *type*] [, *source*] [, *ContentLocation*] [, *ContentBase*])

objAttach

On successful return, contains the new Attachment object.

collAttachments

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.

type

Optional. Long. The type of attachment; either **CdoFileData** or **CdoEmbeddedMessage**. The default value is **CdoFileData**.

source

Optional. String or Object. The path and file name of the file containing the data for the attachment, or the Message object 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.

ContentLocation

Optional. String. The content location header for the appropriate body part of a MIME message attachment.

ContentBase

Optional. String. The content base header for the appropriate body part of a MIME message attachment.

Remarks

The *name*, *type*, *source*, *ContentLocation*, and *ContentBase* parameters correspond to the **Name**, **Type**, **Source**, **ContentLocation**, and **ContentBase** 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 it 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

CdoFileData

Value of *source* parameter

Specifies a full path and file name that contains the data for the attachment, for example C:\DOCUMENT\BUDGET.XLS. Must be supplied with the **Add** method. The data is read into the attachment.

CdoEmbeddedMessage

Specifies the [Message](#) object to be embedded. The message is copied into the attachment.

If the *type* parameter is set to **CdoEmbeddedMessage**, any setting of the *name* parameter is ignored in the current version of CDO for NTS. If the **MessageFormat** property of the parent Message object is set to **CdoMime**, the embedded message's **Subject** property is used for the attachment name. If the parent message's **MessageFormat** is **CdoText**, CDO for NTS randomly creates an attachment name in the format ATT00099.EML.

You should set the message's **MessageFormat** to **CdoMime** if you are running with the Microsoft® Exchange Server and plan to add attachments of type **CdoEmbeddedMessage**. This allows the greatest flexibility in opening and reading such an attachment.

The attachment is saved in persistent storage when you call the **Send** method on the Message object containing the Attachments collection.

The **Add** method returns **CdoE_NO_ACCESS** when called on a message in the Inbox.

The CDO for NTS Library does not permit any modifications to messages in the Inbox, other than deleting the entire message. Prohibited modifications include adding, deleting, or modifying any attachment; adding, deleting, or modifying any recipient; and modifying any message property, even one with read/write access.

Count Property (CDONTS Attachments Collection) Group

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

Syntax

collAttachments.**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  
' collAttachments is an Attachments collection  
x = Util_SmallCollectionCount(collAttachments)
```

```
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_amsmtp  
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 messages 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 (CDONTS Attachments Collection) Group

The **Delete** method removes all the Attachment objects from the Attachments collection.

Syntax

collAttachments.**Delete**()

Remarks

The **Delete** method performs an irreversible operation on the collection. It calls **Release** on the collection's reference to every Attachment object. If you have another reference to an attachment, you can still access its properties and methods, but you can never again associate it with any collection because the **Add** method always creates a new object. You should **Set** your reference variable either to **Nothing** or to another attachment.

The final **Release** on each Attachment object takes place when you call **Delete** if you had no other reference, or when you assign your reference variable to **Nothing**. At this point the object is removed from memory. Attempted access to a released object results in an error return of **CdoE_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 **Send** or **Delete** method on the Message object containing the Attachments collection. A permanently deleted member cannot be recovered. However, the collection itself is still valid, and you can **Add** new members to it.

The CDO for NTS Library does not permit any modifications to messages in the Inbox, other than deleting the entire message. Prohibited modifications include adding, deleting, or modifying any attachment; adding, deleting, or modifying any recipient; and modifying any message property, even one with read/write access.

Item Property (CDONTS Attachments Collection) Group

The **Item** property returns a single [Attachment](#) object from the Attachments collection. Read-only.

Syntax

collAttachments.**Item**(*index*)

index

Long. An integer ranging from 1 to *collAttachments*.**Count**.

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

Data Type

Object (Attachment)

Remarks

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

The **Item**(*index*) syntax selects an arbitrary Attachment object within the Attachments collection.

Although the **Item** property itself is read-only, the [Attachment](#) object it returns can be accessed in the normal manner, and its properties retain their respective read/write or read-only accessibility.

Example

This code fragment shows the [Count](#) and **Item** properties working together to traverse the collection:

```
' from Util_SmallCollectionCount(collAttachments As Object)
Dim strItemName(100) as String
Dim i As Integer ' loop counter
' error handling omitted from this fragment ...
For i = 1 To collAttachments.Count Step 1
    strItemName(i) = collAttachments.Item(i).Name
    ' or = collAttachments(i) since Item and Name are default properties
    If 100 = i Then ' max size of string array
        Exit Function
    End If
Next i
```

Folder Object (CDONTS Library)

The Folder object represents a folder or container in a message store.

At a Glance

Specified in type library: CDONTS.DLL
First available in: CDO for NTS Library version 1.2
Parent objects: [Session](#)
Child objects: [Messages](#) collection
Default property: (none)

Properties

| Name | Available since version | Type | Access |
|-----------------------------|-------------------------|----------------------------|-----------|
| Application | 1.2 | String | Read-only |
| Class | 1.2 | Long | Read-only |
| Messages | 1.2 | Messages collection object | Read-only |
| Name | 1.2 | String | Read-only |
| Parent | 1.2 | Session object | Read-only |
| Session | 1.2 | Session object | Read-only |

Methods

(None.)

Remarks

The CDO for NTS Library does not permit any modifications to messages in the Inbox, other than deleting the entire message. Prohibited modifications include adding, deleting, or modifying any attachment; adding, deleting, or modifying any recipient; and modifying any message property, even one with read/write access.

Messages Property (CDONTS Folder Object)

Group

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

Syntax

objFolder.**Messages**

Data Type

Object (Messages collection)

Remarks

Although the **Messages** property itself is read-only, the collection it returns can be accessed in the normal manner through its **Add** and **Delete** methods, and the properties on its member Message objects retain their respective read/write or read-only accessibility.

The CDO for NTS Library does not permit any modifications to messages in the Inbox, other than deleting the entire message. Prohibited modifications include adding, deleting, or modifying any attachment; adding, deleting, or modifying any recipient; and modifying any message property, even one with read/write access.

Example

This code fragment shows how a Messages collection of Message objects is obtained from a folder which is in turn obtained from a Session object:

```
Dim objInbox as Folder
Dim collInMessages as Messages
' assume valid Session object
Set objInbox = objSession.GetDefaultFolder(CdoDefaultFolderInbox)
Set collInMessages = objInbox.Messages
```


Name Property (CDONTS Folder Object)

Group

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

Syntax

objFolder.Name

Data Type

String

Example

```
Dim objInbox As Folder ' assume valid
Dim objOutbox As Folder ' assume valid
' assume valid Session object
Set objInbox = objSession.GetDefaultFolder(CdoDefaultFolderInbox)
Set objOutbox = objSession.GetDefaultFolder(CdoDefaultFolderOutbox)
MsgBox "Inbox name = " & objInbox.Name
MsgBox "Outbox name = " & objOutbox.Name
```

Message Object (CDONTS Library)

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

At a Glance

| | |
|----------------------------|---|
| Specified in type library: | CDONTS.DLL |
| First available in: | CDO for NTS Library version 1.2 |
| Parent objects: | <u>Messages</u> collection |
| Child objects: | <u>AddressEntry</u> (only obtainable through Sender property of Message object)
<u>Attachments</u> collection
<u>Recipients</u> collection |
| Default property: | <u>Subject</u> |

No modifications are permitted to messages in the Inbox, except for message deletion. Therefore, all properties on a Message object are treated as read-only when the message is in the Inbox. When the access is described as read/write in the following table, that access applies only to messages in the Outbox.

Properties

| Name | Available since version | Type | Access |
|-------------------------------|-------------------------|--|------------|
| <u>Application</u> | 1.2 | String | Read-only |
| <u>Attachments</u> | 1.2 | Attachment object or Attachments collection object | Read-only |
| <u>Class</u> | 1.2 | Long | Read-only |
| <u>ContentBase</u> | 1.2 | String | Read/write |
| <u>ContentID</u> | 1.2 | String | Read-only |
| <u>ContentLocation</u> | 1.2 | String | Read/write |
| <u>HTMLText</u> | 1.2 | IStream object or String | Read/write |
| <u>Importance</u> | 1.2 | Long | Read/write |
| <u>MessageFormat</u> | 1.2 | Long | Write-only |
| <u>Parent</u> | 1.2 | Messages collection object | Read-only |
| <u>Recipients</u> | 1.2 | Recipient object or Recipients collection object | Read-only |
| <u>Sender</u> | 1.2 | AddressEntry object | Read-only |
| <u>Session</u> | 1.2 | Session object | Read-only |
| <u>Size</u> | 1.2 | Long | Read-only |

| | | | |
|----------------------------|-----|---------------------------------|------------|
| <u>Subject</u> | 1.2 | String | Read/write |
| <u>Text</u> | 1.2 | IStream object or String | Read/write |
| <u>TimeReceived</u> | 1.2 | Variant (vbDate format) | Read-only |
| <u>TimeSent</u> | 1.2 | Variant (vbDate format) | Read-only |

Methods

| Name | Available since version | Parameters |
|----------------------|--------------------------------|-------------------|
| <u>Delete</u> | 1.2 | (none) |
| <u>Send</u> | 1.2 | (none) |

Remarks

Microsoft® Visual Basic® programmers can create new Message objects using the Messages collection's **Add** method.

A message can be obtained from its parent Messages collection using the collection's **Item** property. To get to the Messages collection in a folder, use the Folder object's **Messages** property.

The CDO for NTS Library does not permit any modifications to messages in the Inbox, other than deleting the entire message. Prohibited modifications include adding, deleting, or changing any attachment; adding, deleting, or changing any recipient; and writing to any message property, even one with read/write access.

Attachments Property (CDONTS Message Object) Group

The **Attachments** property returns a single Attachment object or an Attachments collection object. Read-only.

Syntax

Set *collAttachments* = *objMessage*.**Attachments**
Set *objAttach* = *objMessage*.**Attachments**(*index*)

collAttachments

Object. An Attachments collection object.

objMessage

Object. The Message object.

objAttach

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.

Data Type

Object (Attachment or Attachments collection)

Remarks

You can change individual Attachment objects within the Attachments collection, **Add** them to the collection, and **Delete** them from the collection.

Although the **Attachments** property itself is read-only, the collection it returns can be accessed in the normal manner through its **Add** and **Delete** methods, and the properties on its member Attachment objects retain their respective read/write or read-only accessibility.

The CDO for NTS Library does not permit any modifications to messages in the Inbox, other than deleting the entire message. Prohibited modifications include adding, deleting, or modifying any attachment; adding, deleting, or modifying any recipient; and modifying any message property, even one with read/write access.

Example

This code fragment uses the **Attachments** property to retrieve an attachment of the message:

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

ContentBase Property (CDONTS Message Object) Group

The **ContentBase** property returns or sets the Content-Base header of a MIME (Multipurpose Internet Mail Extensions) message body. Read/write (read-only for a message in the Inbox).

Syntax

objMessage.**ContentBase**

Data Type

String

Remarks

The **ContentBase** property is used for MHTML (MIME HTML) support. It represents the Content-Base header for the appropriate MIME body part.

For more information on MHTML, see the RFC 2110 document.

ContentID Property (CDONTS Message Object)

Group

The **ContentID** property returns the Content-ID header of a MIME (Multipurpose Internet Mail Extensions) message body. Read-only.

Syntax

objAttach.**ContentID**

Data Type

String

Remarks

The **ContentID** property is used for MHTML (MIME HTML) support. It represents the Content-ID header for the appropriate MIME body part.

The CDO for NTS Library does not generate Content-ID headers for outgoing messages or attachments. This property is provided to inspect an incoming message for Content-ID headers.

For more information on MHTML, see the RFC 2110 document.

ContentLocation Property (CDONTS Message Object) Group

The **ContentLocation** property returns or sets the Content-Location header of a MIME (Multipurpose Internet Mail Extensions) message body. Read/write (read-only for a message in the Inbox).

Syntax

objMessage.**ContentLocation**

Data Type

String

Remarks

The **ContentLocation** property is used for MHTML (MIME HTML) support. It represents the Content-Location header for the appropriate MIME body part.

For more information on MHTML, see the RFC 2110 document.

Delete Method (CDONTS Message Object)

Group

The **Delete** method removes the Message object from the Messages collection.

Syntax

objMessage.Delete()

Remarks

The **Delete** method performs an irreversible operation on the collection. It calls **Release** on the collection's reference to the Message object. If you have another reference to the message, you can still access its properties and methods, but you can never again associate it with any collection because the **Add** method always creates a new object. You should **Set** your reference variable either to **Nothing** or to another message.

The final **Release** on the Message object takes place when you call **Delete** if you had no other reference, or when you assign your reference variable to **Nothing**. At this point the object is removed from memory. Attempted access to a released object results in an error return of **CdoE_INVALID_OBJECT**.

The action of the **Delete** method is permanent, and the Message object cannot be restored to the collection. Before calling **Delete**, your application can prompt the user to verify whether the message should be permanently deleted.

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.

The CDO for NTS Library does not permit any modifications to messages in the Inbox, other than deleting the entire message. Prohibited modifications include adding, deleting, or modifying any attachment; adding, deleting, or modifying any recipient; and modifying any message property, even one with read/write access.

Example

This code fragment illustrates the two situations previously explained. The **Set** statement calls **AddRef** on the first Message object. That reference survives the call to **Delete** and has to be reassigned. The second Message object is deleted without creating another reference, and no other action is necessary.

```
' assume valid Folder object (only Inbox meaningful in this context)
Set objMessage = objFolder.Messages.Item(1)
objMessage.Delete ' still have a reference from Set statement
' ... other operations on objMessage possible but pointless ...
Set objMessage = Nothing ' necessary to remove reference
' ...
objFolder.Messages.Item(2).Delete ' no reference to remove
```


HTMLText Property (CDONTS Message Object)

Group

The **HTMLText** property returns or sets the Hypertext Markup Language (HTML) representation of the message's text. Read/write (read-only for a message in the Inbox).

Syntax

objMessage.HTMLText

Data Type

Object (**IStream**) or String

Remarks

The text is the principal content of an interpersonal message, typically displayed to each recipient as an immediate result of opening the message. It specifically excludes various other message properties such as **Subject**, **Attachments**, and **Recipients**.

The text of a message is represented by its **HTMLText** and **Text** properties. The CDO for NTS Library always keeps these two properties in synchronization with each other. A sending client can set either property and the other is automatically computed. A receiving client can read either property depending on its content type preference.

Only C/C++ and Java programs can use an **IStream** object for this property. They should pass an **IUnknown** object that returns an **IStream** interface in response to **QueryInterface**. Microsoft® Visual Basic® supports the **IDispatch** interface and not **IUnknown**, so it cannot use an **IStream** object.

The maximum size of the text can be limited by the tool that you use to manipulate string variables (for example, Visual Basic).

Importance Property (CDONTS Message Object) Group

The **Importance** property returns or sets the importance of the message. Read/write (read-only for a message in the Inbox).

Syntax

objMessage.Importance

Data Type

Long

Remarks

The following values are defined:

| Constant | Value | Description |
|------------------|-------|-----------------------------|
| CdoLow | 0 | Low importance |
| CdoNormal | 1 | Normal importance (default) |
| CdoHigh | 2 | High importance |

Example

This code fragment sets the importance of a message as high:

```
' assume valid Outbox folder object from GetDefaultFolder method
Set objMessage = objOutbox.Messages.Add
' ... check here to verify the message was created ...
objMessage.Subject = "I'm locked in my office"
objMessage.Text = "Somebody please get the key and let me out."
objMessage.Importance = CdoHigh
objMessage.Send
```

See Also

[Send Method \(Message Object\)](#)

MessageFormat Property (CDONTS Message Object) Group

The **MessageFormat** property sets the encoding format of the message. Write-only.

Syntax

objMessage.**MessageFormat**

Data Type

Long

Remarks

The **MessageFormat** property determines how a message is encoded. The following values are defined:

| MessageFormat setting | Value | Description |
|-----------------------|-------|---|
| CdoMime | 0 | The message is in MIME format. |
| CdoText | 1 | The message is in uninterrupted plain text. |

The **MessageFormat** property is not used on incoming messages in the Inbox.

You should set **MessageFormat** to **CdoMime** if you are running with the Microsoft® Exchange Server and plan to attach embedded messages to this message. This allows the greatest flexibility in opening and reading such an attachment.

The **MessageFormat** property defaults to the current setting of the **MessageFormat** property of the Session object.

Recipients Property (CDONTS Message Object)

Group

The **Recipients** property returns a single Recipient object or a Recipients collection object. Read-only.

Syntax

Set *collRecips* = *objMessage*.**Recipients**

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

collRecips

Object. A Recipients collection object.

objMessage

Object. The Message object.

objRecip

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 (Recipient or Recipients collection)

Remarks

You can change individual Recipient objects within the Recipients collection, **Add** them to the collection, and **Delete** them from the collection.

The CDO for NTS Library does not permit any modifications to messages in the Inbox, other than deleting the entire message. Prohibited modifications include adding, deleting, or modifying any attachment; adding, deleting, or modifying any recipient; and modifying any message property, even one with read/write access.

Example

This code fragment uses a loop to create a copy of every valid recipient of the original message *objMessage* in the copy message *objCopyItem*. For each copied recipient, it also copies important properties from the original.

```
For i = 1 To objMessage.Recipients.Count Step 1
    Set objRecip = objMessage.Recipients.Item(i)
    If Not objRecip Is Nothing Then
        Set objCopyRecip = objCopyItem.Recipients.Add
        If objCopyRecip Is Nothing Then
            MsgBox "Unable to create recipient in message copy"
            Exit Function
        End If
        ' Now copy the most important properties
        objCopyRecip.Address = objRecip.Address
        objCopyRecip.Name = objRecip.Name
        objCopyRecip.Type = objRecip.Type
    End If
Next i
```

Send Method (CDONTS Message Object)

Group

The **Send** method sends the message to the recipients through the messaging system.

Syntax

objMessage.Send()

objMessage

Required. The Message object.

Remarks

The **Send** method saves all changes to the message in the messaging system and moves the message to the current user's Outbox folder. Messaging systems retrieve messages from the Outbox and transport them to the recipients. After it is transported, a message is removed from the Outbox and deleted.

You must compose your new messages in your Outbox. The Send method only deals with messages located in the Outbox and returns **CdoE_NO_ACCESS** for any attempt to create a message in the Inbox.

The **Send** method invalidates the composed 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 sent message results in a return of **CdoE_INVALID_OBJECT**.

Sender Property (CDONTS Message Object)

Group

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 (AddressEntry)

Example

This code fragment displays the name of the sender of a message:

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

Size Property (CDONTS Message Object)

Group

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 **Send** operation. Note that not all message stores support this property.

Subject Property (CDONTS Message Object)

Group

The **Subject** property returns or sets the subject of the message as a string. Read/write (read-only for a message in the Inbox).

Syntax

objMessage.**Subject**

The **Subject** property is the default property of a Message object, meaning that *objMessage* is syntactically equivalent to *objMessage*.**Subject** in Microsoft® Visual Basic® code.

Data Type

String

Remarks

You can set the **Subject** property to an empty string, although that limits the ability of the recipients to sort and filter received messages.

Example

This code fragment sets the subject of a message:

```
Dim objMessage As Message ' assume valid message  
objMessage.Subject = "Test message"
```

See Also

[Text Property \(Message Object\)](#)

Text Property (CDONTS Message Object)

Group

The **Text** property returns or sets the plain text representation of the message's text. Read/write (read-only for a message in the Inbox).

Syntax

objMessage.Text

Data Type

Object (**IStream**) or String

Remarks

The text is the principal content of an interpersonal message, typically displayed to each recipient as an immediate result of opening the message. It specifically excludes various other message properties such as **Subject**, **Attachments**, and **Recipients**.

The text of a message is represented by its **HTMLText** and **Text** properties. The CDO for NTS Library always keeps these two properties in synchronization with each other. A sending client can set either property and the other is automatically computed. A receiving client can read either property depending on its content type preference.

Only C/C++ and Java programs can use an **IStream** object for this property. They should pass an **IUnknown** object that returns an **IStream** interface in response to **QueryInterface**. Microsoft® Visual Basic® supports the **IDispatch** interface and not **IUnknown**, so it cannot use an **IStream** object.

The maximum size of the text can be limited by the tool that you use to manipulate string variables (for example, Visual Basic).

Example

This code fragment sets the text of a message:

```
Dim objMessage As Message ' assume valid message
objMessage.Text = "Text of test message."
```

TimeReceived Property (CDONTS Message Object) Group

The **TimeReceived** property returns the date and time the message was received as a **vbDate** variant data type. Read-only.

Syntax

objMessage.**TimeReceived**

Data Type

Variant (**vbDate** format)

Remarks

If the time of reception is not available, the **TimeReceived** property returns the date and time the message was sent.

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

Example

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

```
' from the sample function Message_TimeSentAndReceived
' verify that objMessage is valid, then ...
With objMessage
    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 (CDONTS Message Object)

Group

The **TimeSent** property returns the date and time the message was sent as a **vbDate** variant data type. Read-only.

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.

Example

See the example for the Message object's **TimeReceived** property.

Messages Collection Object (CDONTS Library)

The Messages collection object contains zero or more Message objects.

At a Glance

| | |
|----------------------------|---------------------------------|
| Specified in type library: | CDONTS.DLL |
| First available in: | CDO for NTS Library version 1.2 |
| Parent objects: | Folder |
| Child objects: | Message |
| Default property: | Item |

Properties

| Name | Available since version | Type | Access |
|-----------------------------|-------------------------|----------------|-----------|
| Application | 1.2 | String | Read-only |
| Class | 1.2 | Long | Read-only |
| Count | 1.2 | Long | Read-only |
| Item | 1.2 | Message object | Read-only |
| Parent | 1.2 | Folder object | Read-only |
| Session | 1.2 | Session object | Read-only |

Methods

| Name | Available since version | Parameters |
|-----------------------------|-------------------------|---|
| Add | 1.2 | (optional) <i>subject</i> as String ,
(optional) <i>text</i> as Object or String ,
(optional) <i>importance</i> as Long |
| Delete | 1.2 | (none) |
| GetFirst | 1.2 | (none) |
| GetLast | 1.2 | (none) |
| GetNext | 1.2 | (none) |
| GetPrevious | 1.2 | (none) |

Remarks

The order that messages are returned by **GetFirst**, **GetLast**, **GetNext**, and **GetPrevious** is not predictable. The best programming approach to use with unsorted collections is to assume that the access functions are able to access all messages within the collection, but that the order of the objects is not defined.

The CDO for NTS Library does not permit any modifications to messages in the Inbox, other than deleting the entire message. Prohibited modifications include adding, deleting, or modifying any attachment; adding, deleting, or modifying any recipient; and modifying any message property, even

one with read/write access.

Add Method (CDONTS Messages Collection)

Group

The **Add** method creates and returns a new Message object in the Messages collection.

Syntax

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

objMessage

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

collMessages

Required. The Messages collection object.

subject

Optional. String. The subject line for the message.

text

Optional. **IStream** object or String. The text of the message.

importance

Optional. Long. The importance associated with the message.

Remarks

The *subject*, *text*, and *importance* parameters correspond to the **Subject**, **Text**, and **Importance** properties on the Message object.

You must create all new messages in the Outbox folder.

Only C/C++ and Java programs can use an **IStream** object for the *text* parameter. They should pass an **IUnknown** object that returns an **IStream** interface in response to **QueryInterface**. Microsoft® Visual Basic® supports the **IDispatch** interface and not **IUnknown**, so it cannot use an **IStream** object.

Example

This code fragment replies to an original message:

```
' from the sample function Util_ReplyToConversation
Set objOutbox = objSession.GetDefaultFolder(CdoDefaultFolderOutbox)
Set objNewMsg = objOutbox.Messages.Add
' verify objNewMsg created successfully ... then supply properties
Set objSenderAE = objOriginalMsg.Sender ' sender as AddressEntry
With objNewMsg
    .Text = "Here is a reply to your message." ' new text
    .Subject = objOriginalMsg.Subject ' copy original properties
    Set objRecip = .Recipients.Add(name:=objSenderAE.Name, _
        type:=CdoTo, _
        address:=objSenderAE.Type & ":" & objSenderAE.Address)
' ** the preceding line is not strictly necessary with CDO for NTS,
' ** but it allows this code to run with CDO for Exchange, where
' ** Recipient.Address requires a FULL address concatenated from
' ** AddressEntry.Type, ":", and AddressEntry.Address
    .Send
End With
```

Count Property (CDONTS Messages Collection)

Group

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

collMessages.**Count**

Data Type

Long

Remarks

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 Messages ' assume collection already provided
Dim objMessage As Message
If collMessages Is Nothing Then
    MsgBox "Messages collection object is invalid"
    ' Exit
Elseif 0 = collMessages.Count Then ' collection is empty
    MsgBox "No messages in collection"
    ' Exit
End If
' look for message about "Bonus" in collection
For i = 1 To collMessages.Count Step 1
    Set objMessage = collMessages.Item(i)
    ' or collMessages(i) since Item is default property
    If objMessage Is Nothing Then ' end of collection
        MsgBox "No such message found in collection"
        Exit For
    Elseif 0 = StrComp(objMsg.Subject, "Bonus") Then
        ' or objMessage since Subject is default property
        MsgBox "Desired message is at index " & i
        Exit For
    End If
Next i
```

Delete Method (CDONTS Messages Collection)

Group

The **Delete** method removes all the Message objects from the Messages collection.

Syntax

collMessages.Delete()

Remarks

The **Delete** method performs an irreversible operation on the collection. It calls **Release** on the collection's reference to every Message object. If you have another reference to a message, you can still access its properties and methods, but you can never again associate it with any collection because the **Add** method always creates a new object. You should **Set** your reference variable either to **Nothing** or to another item.

The final **Release** on each Message object takes place when you call **Delete** if you had no other reference, or when you assign your reference variable to **Nothing**. At this point the object is removed from memory. Attempted access to a released object results in an error return of **CdoE_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 is still valid, and you can **Add** new members to it.

GetFirst Method (CDONTS Messages Collection) Group

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

Syntax

Set *objMessage* = *collMessages*.**GetFirst**()

objMessage

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

collMessages

Required. The Messages collection object.

GetLast Method (CDONTS Messages Collection)

Group

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

Syntax

Set *objMessage* = *collMessages*.**GetLast**()

objMessage

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

collMessages

Required. The Messages collection object.

GetNext Method (CDONTS Messages Collection) Group

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* = *collMessages*.**GetNext**()

objMessage

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

collMessages

Required. The Messages collection object.

Remarks

If the **GetFirst** method has not been called since the Messages collection was initialized, the behavior of the **GetNext** method is not defined. This can produce unexpected results if the collection is reinitialized with a **Set** statement in every iteration of a loop. The recommended procedure is to **Set** an explicit variable for the collection before entering the loop.

If the collection is empty, calling **GetNext** can produce unexpected results even if **GetFirst** has been called.

GetPrevious Method (CDONTS Messages Collection) Group

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* = *collMessages*.**GetPrevious**()

objMessage

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

collMessages

Required. The Messages collection object.

Remarks

If the **GetLast** method has not been called since the Messages collection was initialized, the behavior of the **GetPrevious** method is not defined. This can produce unexpected results if the collection is reinitialized with a **Set** statement in every iteration of a loop. The recommended procedure is to **Set** an explicit variable for the collection before entering the loop.

If the collection is empty, calling **GetPrevious** can produce unexpected results even if **GetLast** has been called.

Item Property (CDONTS Messages Collection)

Group

The **Item** property returns a single Message object from the Messages collection. Read-only.

Syntax

collMessages.**Item**(*index*)

index

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

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

Data Type

Object (Message)

Remarks

The **Item**(*index*) syntax returns the Message object at the indicated position in the collection. It can be used in an indexed loop, such as the **For ... Next** construction in Visual Basic. The first item in the collection has an index of 1.

Although the **Item** property itself is read-only, the Message object it returns can be accessed in the normal manner, and its properties retain their respective read/write or read-only accessibility.

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

NewMail Object (CDONTS Library)

The NewMail object provides for sending a message with very few lines of code.

At a Glance

| | |
|----------------------------|---------------------------------|
| Specified in type library: | CDONTS.DLL |
| First available in: | CDO for NTS Library version 1.2 |
| Parent objects: | (none) |
| Child objects: | (none) |
| Default property: | <u>Value</u> |

Properties

| Name | Available since version | Type | Access |
|-------------------------------|-------------------------|---------------------------------|------------|
| <u>Bcc</u> | 1.2 | String | Write-only |
| <u>Body</u> | 1.2 | IStream object or String | Write-only |
| <u>BodyFormat</u> | 1.2 | Long | Write-only |
| <u>Cc</u> | 1.2 | String | Write-only |
| <u>ContentBase</u> | 1.2 | String | Write-only |
| <u>ContentLocation</u> | 1.2 | String | Write-only |
| <u>From</u> | 1.2 | String | Write-only |
| <u>Importance</u> | 1.2 | Long | Write-only |
| <u>MailFormat</u> | 1.2 | Long | Write-only |
| <u>Subject</u> | 1.2 | String | Write-only |
| <u>To</u> | 1.2 | String | Write-only |
| <u>Value</u> | 1.2 | String | Write-only |
| <u>Version</u> | 1.2 | String | Read-only |

Methods

| Name | Available since version | Parameters |
|--------------------------|-------------------------|---|
| <u>AttachFile</u> | 1.2 | <i>Source</i> as Object or String ,
(optional) <i>FileName</i> as String ,
(optional) <i>EncodingMethod</i> as Long |
| <u>AttachURL</u> | 1.2 | <i>Source</i> as Object or String ,
<i>ContentLocation</i> as String ,
(optional) <i>ContentBase</i> as String ,
(optional) <i>EncodingMethod</i> as Long |
| <u>Send</u> | 1.2 | (optional) <i>From</i> as String ,
(optional) <i>To</i> as String ,
(optional) <i>Subject</i> as String ,
(optional) <i>Body</i> as Object or String , |

Remarks

The NewMail object is not built on the normal API architecture. It is meant for rapid generation of notification mail by an automated process running in the Microsoft® Windows NT® Server. No user interface is supplied, and no interaction with human users is expected during the generation and sending of the message. Therefore the NewMail object's properties are not designed to be read back and inspected. With the sole exception of Version, they can only be written.

The NewMail object is self-contained and does not expose any of the Properties Common to All CDO for NTS Library Objects.

Attachments and recipients, once added to the NewMail object, cannot be removed, and the NewMail object itself cannot be deleted. When the Send method completes successfully, the NewMail object is invalidated but not removed from memory. The programmer should **Set** the invalid object to **Nothing** to remove it from memory, or reassign it to another NewMail object. Attempted access to a sent NewMail object results in a return of **CdoE_INVALID_OBJECT**.

The NewMail object does not belong to the hierarchy encompassing the other CDO for NTS Library objects. It cannot access, nor can it be accessed from, any of the other objects. Like the Session object, it is considered a top-level object and is created directly from a Microsoft® Visual Basic® program. Its ProgID is CDONTS.NewMail. This code fragment creates a NewMail object through early binding:

```
Dim objNewMail As CDONTS.NewMail
Set objNewMail = CreateObject("CDONTS.NewMail")
```

The main advantage of the NewMail object is the ease and simplicity with which you can generate and send a message. You do not have to log on to a session nor deal with a folder or a messages collection. You have only to create the NewMail object, send it, and **Set** it to **Nothing**. You can supply critical information in the parameters of the Send method. In many cases you only need three lines of code:

```
Set objNewMail = CreateObject("CDONTS.NewMail")
objNewMail.Send "me@company.com", "you@company.com", "Hello", _
    "I sent this in 3 statements!", 0 ' low importance
Set objNewMail = Nothing ' canNOT reuse it for another message
```

Including an attachment can add as little as one statement to your code, because you can pass information in the parameters of the AttachFile method:

```
Set objNewMail = CreateObject("CDONTS.NewMail")
objNewMail.AttachFile("\\server\schedule\sched.xls", "SCHED.XLS")
objNewMail.Send "Automated Schedule Generator", "you@company.com", _
    "Schedule", "Here's the latest master schedule", 0
Set objNewMail = Nothing
```

AttachFile Method (CDONTS NewMail Object)

Group

The **AttachFile** method adds an attachment to the message by reading a file.

Syntax

objNewMail.**AttachFile**(*Source* [, *FileName*] [, *EncodingMethod*])

objNewMail

Required. This NewMail object.

Source

Required. **IStream** object or String. The full path and file name of the file to be attached to the message, or a pointer to an **IStream** object containing the file data.

FileName

Optional. String. The file name to appear in the attachment's placeholder in the message. If *FileName* is not supplied, the file name from the *Source* parameter is used.

EncodingMethod

Optional. Long. The manner of encoding the attachment. The following values are possible:

| EncodingMethod setting | Value | Description |
|-------------------------------|--------------|---|
| CdoEncodingUUencode | 0 | The attachment is to be in UUEncode format (default). |
| CdoEncodingBase64 | 1 | The attachment is to be in base 64 format. |

Remarks

The default value for the *EncodingMethod* parameter can change if you set the **MailFormat** property. If **MailFormat** is set to **CdoMailFormatText**, the default value is **CdoEncodingUUencode**. If **MailFormat** is set to **CdoMailFormatMime**, the default value is **CdoEncodingBase64**. However, if you add an attachment encoded in base 64 format, the value of the **MailFormat** property is automatically set to **CdoMailFormatMime**.

Only C/C++ and Java programs can use an **IStream** object for the *Source* parameter. They should pass an **IUnknown** object that returns an **IStream** interface in response to **QueryInterface**. Microsoft® Visual Basic® supports the **IDispatch** interface and not **IUnknown**, so it cannot use an **IStream** object.

AttachURL Method (CDONTS NewMail Object)

Group

The **AttachURL** method adds an attachment to the message and associates a Uniform Resource Locator (URL) with the attachment.

Syntax

objNewMail.**AttachURL**(*Source*, *ContentLocation* [, *ContentBase*] [, *EncodingMethod*])

objNewMail

Required. This NewMail object.

Source

Required. **IStream** object or String. The full path and file name of the resource to be attached to the message, or a pointer to an **IStream** object containing the file data.

ContentLocation

Required. String. The absolute or relative prefix for the URL that the rendering client can use to reference this attachment.

ContentBase

Optional. String. A base for the URL used to reference this attachment.

EncodingMethod

Optional. Long. The manner of encoding the attachment. The following values are possible:

| EncodingMethod setting | Value | Description |
|-------------------------------|--------------|---|
| CdoEncodingUUencode | 0 | The attachment is to be in UUencode format (default). |
| CdoEncodingBase64 | 1 | The attachment is to be in base 64 format. |

Remarks

You must supply at least the *ContentLocation* parameter to specify a URL for the attachment. If you also supply the *ContentBase* parameter, it is combined with the *ContentLocation* parameter to define the full URL path by which this attachment is to be referenced. For more information on constructing URL paths, see the [ContentLocation](#) property.

The default value for the *EncodingMethod* parameter can change if you set the **MailFormat** property. If **MailFormat** is set to **CdoMailFormatText**, the default value is **CdoEncodingUUencode**. If **MailFormat** is set to **CdoMailFormatMime**, the default value is **CdoEncodingBase64**. However, if you add an attachment encoded in base 64 format, the value of the **MailFormat** property is automatically set to **CdoMailFormatMime**.

Only C/C++ and Java programs can use an **IStream** object for the *Source* parameter. They should pass an **IUnknown** object that returns an **IStream** interface in response to **QueryInterface**. Microsoft® Visual Basic® supports the **IDispatch** interface and not **IUnknown**, so it cannot use an **IStream** object.

Example

This code fragment illustrates the preparation and sending of an MHTML (MIME HTML) message and attachment using the **AttachURL** method:

```
Dim myMail
Set myMail = CreateObject("CDONTS.NewMail")
```

```
HTML = "<!DOCTYPE HTML PUBLIC ""-//IETF//DTD HTML//EN"">" & NL
HTML = HTML & "<html>"
```

```
HTML = HTML & "<head>"
HTML = HTML & "<meta http-equiv=""Content-Type""
HTML = HTML & "content=""text/html; charset=iso-8859-1"">"
HTML = HTML & "<meta name=""GENERATOR"" content=""Microsoft FrontPage 2.0"">"
HTML = HTML & "<title>Exchange CDO Example</title>"
HTML = HTML & "</head>"
HTML = HTML & "<body bgcolor=""#FFFFFF"">"
HTML = HTML & "<p><font size=""6"" face=""Arial Black""><strong>Exchange CDO "
HTML = HTML & "Sample<img src=CDO.gif>"
HTML = HTML & "</strong></font></p>"
HTML = HTML & "<p>CDO for NTS allows an easy way to send mail."
HTML = HTML & "This example shows how the content can be an HTML page"
HTML = HTML & "which allows you to send rich text and inline graphics.</p>"
HTML = HTML & "</body>"
HTML = HTML & "</html>"
```

```
myMail.AttachURL "D:\wwwroot\CDO.gif", "CDO.gif"
myMail.From = "Example@Microsoft.com"
myMail.To = "Someone@Microsoft.com"
myMail.Subject = "Sample Message"
myMail.BodyFormat = 0
myMail.MailFormat = 0
myMail.Body = HTML
myMail.Send
```

Bcc Property (CDONTS NewMail Object)

Group

The **Bcc** property adds to the list of blind copy (Bcc) recipients for the NewMail object. Write-only.

Syntax

objNewMail.**Bcc**

Data Type

String

Remarks

The value you use to set the **Bcc** property can represent a single recipient or a list of recipients. Each recipient must be represented by a full messaging address:

```
"useraddress@company.com"
```

Multiple recipients on the list are separated by semicolons:

```
"user1@company1.com;user2@company2.com;user3@company3.com"
```

Body Property (CDONTS NewMail Object)

Group

The **Body** property sets the text of the NewMail object. Write-only.

Syntax

objNewMail.**Body**

Data Type

Object (**IStream**) or String

Remarks

The **Body** property can contain either plain text or HTML. The **BodyFormat** property should be set to indicate whether or not the **Body** property includes any HTML.

Only C/C++ and Java programs can use an **IStream** object for this property. They should pass an **IUnknown** object that returns an **IStream** interface in response to **QueryInterface**. Microsoft® Visual Basic® supports the **IDispatch** interface and not **IUnknown**, so it cannot use an **IStream** object.

BodyFormat Property (CDONTS NewMail Object) Group

The **BodyFormat** property sets the text format of the NewMail object. Write-only.

Syntax

objNewMail.**BodyFormat**

Data Type

Long

Remarks

BodyFormat can contain exactly one of the following values:

| BodyFormat setting | Value | Description |
|--------------------------|-------|--|
| CdoBodyFormatHTML | 0 | The Body property is to include Hypertext Markup Language (HTML). |
| CdoBodyFormatText | 1 | The Body property is to be exclusively in plain text (default value). |

Example

This code fragment shows the usage of the **BodyFormat** property in preparing and sending an HTML message:

```
Dim myMail
Set myMail = CreateObject("CDONTS.NewMail")

HTML = "<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML//EN">" & NL
HTML = HTML & "<html>"
HTML = HTML & "<head>"
HTML = HTML & "<meta http-equiv=""Content-Type""
HTML = HTML & "HTML = HTML & ""content=""text/html; charset=iso-8859-1"">""
HTML = HTML & "<title>Sample NewMail</title>"
HTML = HTML & "</head>"
HTML = HTML & "<body>"
HTML = HTML & "This is a sample message being sent using HTML. <BR></body>"
HTML = HTML & "</html>"

myMail.From = "Example@Microsoft.com"
myMail.To = "Someone@Company.com"
myMail.Subject = "Sample Message"
myMail.BodyFormat = 0
myMail.MailFormat = 0
myMail.Body = HTML
myMail.Send
Set myMail = Nothing
```

Cc Property (CDONTS NewMail Object)

Group

The **Cc** property adds to the list of copy (Cc) recipients for the NewMail object. Write-only.

Syntax

objNewMail.Cc

Data Type

String

Remarks

The value you use to set the **Cc** property can represent a single recipient or a list of recipients. Each recipient must be represented by a full messaging address:

```
"useraddress@company.com"
```

Multiple recipients on the list are separated by semicolons:

```
"user1@company1.com;user2@company2.com;user3@company3.com"
```

ContentBase Property (CDONTS NewMail Object) Group

The **ContentBase** property sets a base for all URLs relating to the NewMail object's message body. Write-only.

Syntax

objNewMail.**ContentBase**

Data Type

String

Remarks

The **ContentBase** property is used for MHTML (MIME HTML) support. It represents the Content-Base header for URLs pertaining to the main body of a MIME message. **ContentBase** corresponds to the **ContentBase** property of a [Message](#) object.

The **ContentBase** property is used in conjunction with the **ContentLocation** property to provide an absolute path for all URLs pertaining to the message body. These include the URL used to reference the message body itself, and also any URLs within HTML tags in the **Body** property of the NewMail object.

ContentBase is meant to be combined with **ContentLocation** to produce an absolute path. For more information on constructing URL paths, see the [ContentLocation](#) property.

For more information on MHTML, see the RFC 2110 document.

ContentLocation Property (CDONTS NewMail Object) Group

The **ContentLocation** property sets an absolute or relative path for all URLs relating to the NewMail object's message body. Write-only.

Syntax

objNewMail.**ContentLocation**

Data Type

String

Remarks

The **ContentLocation** property is used for MHTML (MIME HTML) support. It represents the Content-Location header for URLs pertaining to the main body of a MIME message. **ContentLocation** corresponds to the **ContentLocation** property of a Message object.

If the **ContentLocation** property is set, it is taken as an absolute or relative URL that can be used to refer to the message body of the NewMail object. If the **ContentBase** property is also set, **ContentLocation** is taken as relative and is joined to the base provided by **ContentBase**.

The **Body** property of the NewMail object can also contain HTML tags that use URLs, for example and <FORM>. If the resource for such a URL is not local to the recipient of the NewMail object, you must provide a path to locate the URL on the Internet. One way to do this is to provide the full path in the URL itself:

```
<IMG SRC=HTTP://www.abc.com/graphs/Feb1998/16Feb98/today.gif>
```

It is often more flexible, however, to supply the path externally and put only the resource name in the URL:

```
<IMG SRC=today.gif>
```

When this approach is taken, the **ContentLocation** and **ContentBase** properties can be used to supply the path. **ContentLocation** can contain an absolute path:

```
objNewMail.ContentLocation = "HTTP://www.abc.com/graphs/Feb1998/16Feb98/"
```

or a relative path:

```
objNewMail.ContentLocation = "Feb1998/16Feb98/"
```

If **ContentLocation** contains a relative path, the base URL is supplied in **ContentBase**:

```
objNewMail.ContentBase = "HTTP://www.abc.com/graphs/"
```

When **ContentBase** and **ContentLocation** are combined using standard URL combination rules, the result is an absolute path to the resource referenced by the URL.

Often a resource is included as an attachment to the NewMail object. When this is the case, the *ContentLocation* and *ContentBase* parameters of the **AttachURL** method are used to specify the attachment's URL. They are combined the same way the **ContentLocation** and **ContentBase** properties are combined. The URL specified by the attachment must match the URL requested by the HTML tag in the **Body** property, or the tag is not successfully resolved.

The simplest case of attaching with URLs is when your message body refers to the attachments by their *ContentLocation* parameter values. In this case you do not need either the **ContentLocation** or

the **ContentBase** property, and the attachments only need resource names in their *ContentLocation* parameters. This code fragment sends a weekly review of sales activity with two chart images attached:

```
Set objNewMail = CreateObject("CDONTS.NewMail")
strBody = "<HTML><HEAD></HEAD><BODY>" _
    & "This is the sales chart for the past week:" & vbCrLf _
    & "<IMG SRC=week.gif>" & vbCrLf _
    & "and this is the sales chart for the month:" & vbCrLf _
    & "<IMG SRC=month.gif>" & "</BODY></HTML>"
objNewMail.AttachURL "\\myserver\sales\pastweek.gif", "week.gif"
objNewMail.AttachURL "\\myserver\sales\thismnth.gif", "month.gif"
objNewMail.Send "Automated Sales Report", "you@company.com", _
    "Sales Charts", strBody, 0
Set objNewMail = Nothing ' canNOT reuse it for another message
```

Note that the URL string specified in the *ContentLocation* parameter of the **AttachURL** method does not have to match the file name of the underlying resource. The only requirement is that it match the URL requested by the HTML tag in the message body.

For more information on MHTML, see the RFC 2110 document.

From Property (CDONTS NewMail Object)

Group

The **From** property sets the full messaging address to be used for the sender of this NewMail object. Write-only.

Syntax

objNewMail.**From**

Data Type

String

Remarks

The full messaging address must take the form

"senderaddress@company.com"

You cannot use a semicolon anywhere in the **From** string. Semicolons are reserved for multiple messaging addresses in a list, and the CDO for NTS Library does not permit multiple senders.

Importance Property (CDONTS NewMail Object)

Group

The **Importance** property sets the importance associated with the NewMail object. Write-only.

Syntax

objNewMail.Importance

Data Type

Long

Remarks

The following values are defined:

| Constant | Value | Description |
|------------------|-------|-----------------------------|
| CdoLow | 0 | Low importance |
| CdoNormal | 1 | Normal importance (default) |
| CdoHigh | 2 | High importance |

MailFormat Property (CDONTS NewMail Object)

Group

The **MailFormat** property sets the encoding for the NewMail object. Write-only.

Syntax

objNewMail.**MailFormat**

Data Type

Long

Remarks

MailFormat can contain exactly one of the following values:

| MailFormat setting | Value | Description |
|--------------------------|-------|--|
| CdoMailFormatMime | 0 | The NewMail object is to be in MIME format. |
| CdoMailFormatText | 1 | The NewMail object is to be in uninterrupted plain text (default value). |

The setting of the **MailFormat** property determines the default value for the *EncodingMethod* parameter in the **AttachFile** and **AttachURL** methods. However, if you add an attachment encoded in base 64 format, the value of the **MailFormat** property is automatically set to **CdoMailFormatMime**.

For more information on Multipurpose Internet Mail Extensions (MIME), see the RFC 1341 document.

Send Method (CDONTS NewMail Object)

Group

The **Send** method sends the NewMail object to the specified recipients.

Syntax

objNewMail.**Send**([*From*] [, *To*] [, *Subject*] [, *Body*] [, *Importance*])

objNewMail

Required. This NewMail object.

From

Optional. String. The full messaging address to be identified as the sender.

To

Optional. String. A list of full messaging addresses of recipients. The individual recipient addresses are separated by semicolons.

Subject

Optional. String. The subject line for the message.

Body

Optional. **IStream** object or String. The text of the message.

Importance

Optional. Long. The importance associated with the message.

Remarks

The *From*, *To*, *Subject*, *Body*, and *Importance* parameters correspond to the **From**, **To**, **Subject**, **Body**, and **Importance** properties on the NewMail object.

If both the **To** property and the *To* parameter of the **Send** method are supplied, the NewMail object is sent to all recipients on both lists.

Only C/C++ and Java programs can use an **IStream** object for the *Body* parameter. They should pass an **IUnknown** object that returns an **IStream** interface in response to **QueryInterface**. Microsoft® Visual Basic® supports the **IDispatch** interface and not **IUnknown**, so it cannot use an **IStream** object.

The NewMail object becomes invalid upon successful completion of the **Send** method, and you cannot reuse it for another message. You should **Set** it to **Nothing** to release the memory. Attempted access to a sent NewMail object results in a return of **CdoE_INVALID_OBJECT**.

Example

This code fragment demonstrates a simple usage of the NewMail object and its **Send** method:

```
Dim myMail
Set myMail = CreateObject("CDONTS.NewMail")
myMail.From = "Example@Microsoft.com"
myMail.To = "Someone@company.com"
myMail.Subject = "Sample Message"
myMail.Body = "This is a sample message."
myMail.AttachFile "d:\sample.txt"
myMail.Send
Set myMail = Nothing
```

SetLocaleIDs Method (CDONTS NewMail Object) Group

The **SetLocaleIDs** method sets identifiers that define a messaging user's locale.

Syntax

objNewMail.**SetLocaleIDs**(*CodePageID*)

objNewMail

Required. This NewMail object.

CodePageID

Required. Long. The code page identifier to be used for this messaging user.

Remarks

A locale is the set of features of a messaging user's environment that are dependent on language, country, culture, and conventions. These features include the character selection, the collating sequence and sort order, and the date, time, and currency formats. The **SetLocaleIDs** method sets identifiers that determine the behavior of locale-sensitive operations.

A code page identifier is a long integer specifying the ordered character set to use when displaying text. Information about a code page can be obtained from the Windows NT **GetCPInfo** function.

When you first create a NewMail object it uses the locale values in the user's registry. However, you can change a NewMail object's locale after its creation by calling the its **SetLocaleIDs** method. Such a call does not affect the locale of any other Message or NewMail object, nor does it alter the settings established by a Session object's **SetLocaleIDs** method.

SetLocaleIDs tests the validity of the code page specified by the *CodePageID* parameter before actually setting the locale identifiers. If the code page is not valid, **CdoE_INVALID_PARAMETER** is returned.

Subject Property (CDONTS NewMail Object)

Group

The **Subject** property sets the subject of the NewMail object as a string. Write-only.

Syntax

objNewMail.**Subject**

Data Type

String

Remarks

You can set the **Subject** property to an empty string, although that limits the ability of the recipients to sort and filter received messages.

To Property (CDONTS NewMail Object)

Group

The **To** property adds to the list of principal (To) recipients for the NewMail object. Write-only.

Syntax

objNewMail.To

Data Type

String

Remarks

The value you use to set the **To** property can represent a single recipient or a list of recipients. Each recipient must be represented by a full messaging address:

```
"useraddress@company.com"
```

Multiple recipients on the list are separated by semicolons:

```
"user1@company1.com;user2@company2.com;user3@company3.com"
```

If both the **To** property and the *To* parameter of the **Send** method are supplied, the NewMail object is sent to all recipients on both lists.

Value Property (CDONTS NewMail Object)

Group

The **Value** property sets the value and contents of an additional header for the NewMail object. Write-only.

Syntax

objNewMail.**Value**(*header*) = *strHdrValue*

header

A character string containing the name of the header to be added.

strHdrValue

A character string containing the value of the header to be added.

The **Value** property is the default property of a NewMail object, meaning that *objNewMail(header)* is syntactically equivalent to *objNewMail.Value(header)* in Microsoft® Visual Basic® code.

Data Type

String

Remarks

The **Value** property is used to add one or more headers to the automatically generated headers such as "To", "From", "Subject", and "Date". Possibilities for additional headers include "File", "Keywords", and "Reference".

Certain headers, such as "Reply-To", are widely accepted and used by various messaging systems. If you wish such a header to be recognized by the recipients of the NewMail object, you must be sure that the character string in the header's name exactly matches the accepted string.

In principle you can put any combination of ASCII characters in the string, but some messaging systems may have restrictions on the character set. The safest procedure is to limit the string to alphanumeric characters, dashes, and slashes, and in particular to avoid spaces.

You can set the **Value** property more than once. Each setting generates another header to be included with the existing headers.

Example

This code fragment adds two headers to a NewMail object before sending it:

```
' strBody set in advance for message body
objNewMail.Value("Reply-To") = "The Boss<myemail@mycompany.com>"
objNewMail.Value("Confidential") = "For my direct reports only"
objNewMail.Send "myemail@mycompany.com", _
                "myreports@mycompany.com", _
                "Reorganization", strBody, 2 ' high importance
Set objNewMail = Nothing
```

Version Property (CDONTS NewMail Object)

Group

The **Version** property returns the version of the CDO for NTS Library. Read-only.

Syntax

objNewMail.**Version**

Data Type

String

Remarks

The **Version** property currently returns "1.2.1".

Recipient Object (CDONTS Library)

The Recipient object represents a recipient of a message.

At a Glance

| | |
|----------------------------|---------------------------------|
| Specified in type library: | CDONTS.DLL |
| First available in: | CDO for NTS Library version 1.2 |
| Parent objects: | <u>Recipients</u> collection |
| Child objects: | (none) |
| Default property: | <u>Name</u> |

No modifications are permitted to messages in the Inbox, except for message deletion. Therefore, all properties on a Recipient object are treated as read-only when the grandparent Message object is in the Inbox. When the access is described as read/write in the following table, that access applies only to recipients on messages in the Outbox.

Properties

| Name | Available since version | Type | Access |
|--------------------|-------------------------|------------------------------|------------|
| <u>Address</u> | 1.2 | String | Read/write |
| <u>Application</u> | 1.2 | String | Read-only |
| <u>Class</u> | 1.2 | Long | Read-only |
| <u>Name</u> | 1.2 | String | Read/write |
| <u>Parent</u> | 1.2 | Recipients collection object | Read-only |
| <u>Session</u> | 1.2 | Session object | Read-only |
| <u>Type</u> | 1.2 | Long | Read/write |

Methods

| Name | Available since version | Parameters |
|---------------|-------------------------|------------|
| <u>Delete</u> | 1.2 | (none) |

Address Property (CDONTS Recipient Object)

Group

The **Address** property specifies the full messaging address for the recipient. Read/write (read-only for a recipient on a message in the Inbox).

Syntax

objRecip.**Address**

Data Type

String

Remarks

The Recipient object's **Address** property represents the complete messaging address used by the messaging system.

Example

```
' assume valid Recipient object  
objRecip.Address = "myname@mycompany.com"
```

Delete Method (CDONTS Recipient Object)

Group

The **Delete** method removes the Recipient object from the Recipients collection.

Syntax

objRecip.Delete()

Remarks

The **Delete** method performs an irreversible operation on the collection. It calls **Release** on the collection's reference to the Recipient object. If you have another reference to the recipient, you can still access its properties and methods, but you can never again associate it with any collection because the **Add** method always creates a new object. You should **Set** your reference variable either to **Nothing** or to another recipient.

The final **Release** on the Recipient object takes place when you assign your reference variable to **Nothing**, or when you call **Delete** if you had no other reference. At this point the object is removed from memory. Attempted access to a released object results in an error return of **CdoE_INVALID_OBJECT**.

The effect of the **Delete** operation is not permanent until you use the **Send** or **Delete** method on the Message object to which this recipient belongs.

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.

The CDO for NTS Library does not permit any modifications to messages in the Inbox, other than deleting the entire message. Prohibited modifications include adding, deleting, or modifying any attachment; adding, deleting, or modifying any recipient; and modifying any message property, even one with read/write access.

Example

This code fragment illustrates the two situations previously explained. The **Set** statement calls **AddRef** on the first Recipient object. That reference survives the call to **Delete** and has to be reassigned. The second Recipient object is deleted without creating another reference, and no other action is necessary.

```
' assume valid Message object
Set objRecip = objMessage.Recipients.Item(1)
objRecip.Delete ' still have a reference from Set statement
' ... other operations on objRecip possible but pointless ...
Set objRecip = Nothing ' necessary to remove reference
' ...
objMessage.Recipients.Item(2).Delete ' no reference to remove
```

Name Property (CDONTS Recipient Object)

Group

The **Name** property returns or sets the name of the Recipient object as a string. Read/write (read-only for a recipient on a message in the Inbox).

Syntax

objRecip.**Name**

The **Name** property is the default property of a Recipient object, meaning that *objRecip* is syntactically equivalent to *objRecip*.**Name** in Microsoft® Visual Basic® code.

Data Type

String

Remarks

The **Name** property returns an empty string if it has never been set or cannot be found.

Example

```
Dim strMsg As String
' ... validate objects ... then display
strMsg = "Recipient address = " & objRecip.Address
strMsg = strMsg & "; Recipient name = " & objRecip.Name
MsgBox strMsg ' display recipient parts
```

Type Property (CDONTS Recipient Object)

Group

The **Type** property specifies the recipient type of the Recipient object, that is, whether it is a To, Cc, or Bcc recipient. Read/write (read-only for a recipient on a message in the Inbox).

Syntax

objRecip.Type

Data Type

Long

Remarks

The **Type** property has the following defined values:

| Recipient type | Value | Description |
|----------------|-------|--|
| CdoTo | 1 | The recipient is on the To line (default). |
| CdoCc | 2 | The recipient is on the Cc line. |
| CdoBcc | 3 | The recipient is on the Bcc line. |

See Also

[Address Property \(Recipient Object\)](#)

Recipients Collection Object (CDONTS Library)

The Recipients collection object contains zero or more Recipient objects and specifies the recipients of a message.

At a Glance

| | |
|----------------------------|---------------------------------|
| Specified in type library: | CDONTS.DLL |
| First available in: | CDO for NTS Library version 1.2 |
| Parent objects: | Message |
| Child objects: | Recipient |
| Default property: | Item |

Properties

| Name | Available since version | Type | Access |
|-----------------------------|-------------------------|------------------|-----------|
| Application | 1.2 | String | Read-only |
| Class | 1.2 | Long | Read-only |
| Count | 1.2 | Long | Read-only |
| Item | 1.2 | Recipient object | Read-only |
| Parent | 1.2 | Message object | Read-only |
| Session | 1.2 | Session object | Read-only |

Methods

| Name | Available since version | Parameters |
|------------------------|-------------------------|--|
| Add | 1.2 | (optional) <i>name</i> as String ,
(optional) <i>address</i> as String ,
(optional) <i>type</i> as Long |
| Delete | 1.2 | (none) |

Add Method (CDONTS Recipients Collection)

Group

The **Add** method creates and returns a new Recipient object in the Recipients collection.

Syntax

Set *objRecip* = *collRecips*.**Add**([*name*] [, *address*] [, *type*])

objRecip

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

collRecips

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 full 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 recipient type; the initial value for the new recipient's **Type** property. The following values are valid:

| Recipient type | Value | Description |
|----------------|-------|--|
| CdoTo | 1 | The recipient is on the To line (default). |
| CdoCc | 2 | The recipient is on the Cc line. |
| CdoBcc | 3 | The recipient is on the Bcc line. |

Remarks

The *name*, *address*, and *type* parameters correspond to the Recipient object's **Name**, **Address**, and **Type** properties, respectively.

When no parameters are present, an empty Recipient object is created.

The new recipient is saved in persistent storage when you call the **Send** method on the Message object containing the Recipients collection.

The **Add** method returns **CdoE_NO_ACCESS** when called on a message in the Inbox.

The CDO for NTS Library does not permit any modifications to messages in the Inbox, other than deleting the entire message. Prohibited modifications include adding, deleting, or modifying any attachment; adding, deleting, or modifying any recipient; and modifying any message property, even one with read/write access.

Example

This code fragment adds a recipient to a message using information from an existing valid AddressEntry object

```
' from the sample function "Using Addresses"
' assume valid address entry ID from an existing message
Set objRecip = objNewMessage.Recipients.Add(type:=CdoTo, _
                                           name:=objAdrEnt.Name)
objRecip.Address = objAdrEnt.Type & ":" & objAdrEnt.Address
' ** the preceding line is not strictly necessary with CDO for NTS,
' ** but it allows this code to run with CDO for Exchange, where
```

```
' ** Recipient.Address requires a FULL address concatenated from  
' ** AddressEntry.Type, ":", and AddressEntry.Address  
If objRecip Is Nothing Then  
    MsgBox "Unable to add existing AddressEntry using ID"  
    Exit Function  
End If  
  
objNewMessage.Text = "Expect 1 recipient."  
MsgBox ("Count = " & objNewMessage.Recipients.Count)
```

Count Property (CDONTS Recipients Collection)

Group

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

Syntax

collRecips.Count

Data Type

Long

Example

This code fragment uses the **Count** property as a loop terminator to copy all Recipient objects from one message's Recipients collection to another message's collection. It shows the **Count** and Item properties working together. Note how much more code this requires than copying the Message object's Recipients property from the original message to the copy.

```
' Copy all Recipient objects from one message's collection to another
Dim objMessage, objCopyItem as Message
Dim collRecips as Recipients ' source message Recipients collection
Dim objRecip as Recipient ' individual recipient in target message
' ... verify valid messages ...
Set collRecips = objMessage.Recipients
For i = 1 To collRecips.Count Step 1
    strRecipName = collRecips.Item(i).Name
    ' could be collRecips(i).Name since Item is default property
    If strRecipName <> "" Then
        Set objRecip = objCopyItem.Recipients.Add
        If objRecip Is Nothing Then
            MsgBox "Unable to create recipient in message copy"
            Exit Function
        End If
        objRecip.Name = strRecipName
        objRecip.Address = collRecips.Item(i).Address
        objRecip.Type = collRecips.Item(i).Type
    End If
Next i
```

Delete Method (CDONTS Recipients Collection)

Group

The **Delete** method removes all the Recipient objects from the Recipients collection.

Syntax

collRecips.**Delete**()

Remarks

The **Delete** method performs an irreversible operation on the collection. It calls **Release** on the collection's reference to every Recipient object. If you have another reference to a recipient, you can still access its properties and methods, but you can never again associate it with any collection because the **Add** method always creates a new object. You should **Set** your reference variable either to **Nothing** or to another recipient.

The final **Release** on each Recipient object takes place when you call **Delete** if you had no other reference, or when you assign your reference variable to **Nothing**. At this point the object is removed from memory. Attempted access to a released object results in an error return of **CdoE_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 **Send** or **Delete** method on the Message object containing the Recipients collection. A permanently deleted member cannot be recovered. However, the collection itself is still valid, and you can **Add** new members to it.

The CDO for NTS Library does not permit any modifications to messages in the Inbox, other than deleting the entire message. Prohibited modifications include adding, deleting, or modifying any attachment; adding, deleting, or modifying any recipient; and modifying any message property, even one with read/write access.

Item Property (CDONTS Recipients Collection)

Group

The **Item** property returns a single Recipient object from the Recipients collection. Read-only.

Syntax

collRecips.**Item**(*index*)

index

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

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

Data Type

Object (Recipient)

Remarks

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

Although the **Item** property itself is read-only, the Recipient object it returns can be accessed in the normal manner, and its properties retain their respective read/write or read-only accessibility.

Example

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

```
' list all recipient names in the collection
strRecips = "" ' initialize string
Set collRecips = objMessage.Recipients
Count = collRecips.Count
For i = 1 To Count Step 1
    Set objRecip = collRecips.Item(i) ' or collRecips(i)
    strRecips = strRecips & objRecip.Name & "; "
Next i
MsgBox "Message recipients: " & strRecips
```

Session Object (CDONTS Library)

The Session object contains session-wide settings and options.

At a Glance

| | |
|----------------------------|---------------------------------|
| Specified in type library: | CDONTS.DLL |
| First available in: | CDO for NTS Library version 1.2 |
| Parent objects: | (none) |
| Child objects: | <u>Folder</u> |
| Default property: | <u>Name</u> |

Properties

| Name | Available since version | Type | Access |
|----------------------|-------------------------|-------------------------------|------------|
| <u>Application</u> | 1.2 | String | Read-only |
| <u>Class</u> | 1.2 | Long | Read-only |
| <u>Inbox</u> | 1.2 | Folder object | Read-only |
| <u>MessageFormat</u> | 1.2 | Long | Read/write |
| <u>Name</u> | 1.2 | String | Read-only |
| <u>Outbox</u> | 1.2 | Folder object | Read-only |
| <u>Parent</u> | 1.2 | Object; set to Nothing | Read-only |
| <u>Session</u> | 1.2 | Session object (itself) | Read-only |
| <u>Version</u> | 1.2 | String | Read-only |

Methods

| Name | Available since version | Parameters |
|-------------------------|-------------------------|--|
| <u>GetDefaultFolder</u> | 1.2 | <i>folderType</i> as Long |
| <u>Logoff</u> | 1.2 | (none) |
| <u>LogonSMTP</u> | 1.2 | <i>DisplayName</i> as String ,
<i>Address</i> as String |
| <u>SetLocaleIDs</u> | 1.2 | <i>CodePageID</i> as Long |

Remarks

A Session object is considered a top-level object, meaning it can be created directly from a Microsoft® Visual Basic® program. In the CDO for NTS Library it has a ProgID of CDONTS.Session. This code fragment creates a Session object through early binding:

```
Dim objSession As CDONTS.Session
Set objSession = CreateObject ("CDONTS.Session")
objSession.LogonSMTP ("Display Name", "address@company.com")
```

This code fragment creates a Session object through late binding:

```
Dim objSession As Object
Set objSession = CreateObject ("CDONTS.Session")
objSession.LogonSMTP ("Display Name", "address@company.com")
```

Generally, early binding is preferable, because it enforces type checking and generates more efficient code. Note that you specify the full ProgID "CDONTS.Session" instead of just "Session" in order to distinguish a CDO application from other types of applications available to a Visual Basic program through other object libraries.

In both cases, after you create a new Session object, you call its **LogonSMTP** method to initialize it. No other activities with the CDO for NTS Library are permitted prior to a successful **LogonSMTP** call. The only exception to this rule is the Session object's **SetLocaleIDs** method.

GetDefaultFolder Method (CDONTS Session Object) Group

The **GetDefaultFolder** method returns a Folder object from a message store.

Syntax

Set *objFolder* = *objSession*.**GetDefaultFolder**(*folderType*)

objFolder

On successful return, contains the store's default Folder object of the specified type. When the folder does not exist, **GetDefaultFolder** returns **Nothing**.

objSession

Required. The Session object.

folderType

Required. Long. The folder type. This parameter can have exactly one of the following values:

| folderType setting | Decimal value | Meaning |
|-------------------------------|----------------------|-----------------------------------|
| CdoDefaultFolderInbox | 1 | The CDO for NTS Library's Inbox. |
| CdoDefaultFolderOutbox | 2 | The CDO for NTS Library's Outbox. |

Remarks

The **GetDefaultFolder** method allows you to obtain the message store's default Inbox or Outbox folder. No other folders are accessible through this method.

Example

This code fragment uses the **GetDefaultFolder** method to obtain the Inbox folder from a message store:

```
Dim objSession As Session
Dim objInbox As Folder ' default Inbox or Outbox
Dim collMessages As Messages ' messages in folder

Set objSession = CreateObject ("CDONTS.Session")
objSession.LogonSMTP("My Name", "myaddress@mycompany.com")
Set objInbox = objSession.GetDefaultFolder(CdoDefaultFolderInbox)
If objInbox Is Nothing Then
    Set collMessages = Nothing
    MsgBox "Unable to retrieve default folder"
    Exit Function
Else
    Set collMessages = objInbox.Messages
    MsgBox "Folder set to " & objInbox.Name
End If
```


Inbox Property (CDONTS Session Object)

Group

The **Inbox** property returns a Folder object representing the current messaging user's Inbox folder. Read-only.

Syntax

objSession.**Inbox**

Data Type

Object (Folder)

Logoff Method (CDONTS Session Object)

Group

The **Logoff** method uninitialized the Session object.

Syntax

objSession.**Logoff**()

Remarks

The **Logoff** method terminates all activity on the Session object initialized by the **LogonSMTP** method. You can call **LogonSMTP** on the same Session object again. Attempted access to the Session object before initialization results in a return of **CdoE_NOT_INITIALIZED**.

Objects you create while you are active on a Session object become invalid when you call **Logoff**. You should **Set** your reference variables to **Nothing** for all such objects.

Example

This code fragment logs off from the messaging 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; cannot log off"
End If
```

LogonSMTP Method (CDONTS Session Object)

Group

The **LogonSMTP** method initializes the Session object.

Syntax

objSession.**LogonSMTP**(*DisplayName*, *Address*)

objSession

Required. The Session object.

DisplayName

Required. String. The display name to use for the messaging user logging on, such as "John Q. Doe".

Address

Required. String. The full e-mail address to use for the messaging user logging on, such as "jdoe@company.com".

Remarks

Your application must call **LogonSMTP** before it can use any CDO for NTS Library object, including the Session object. An attempt to access any programming element prior to a successful **LogonSMTP** call results in a **CdoE_NOT_INITIALIZED** error return. The only exception to this rule is the Session object's **SetLocaleIDs** method.

An application using the CDO for NTS Library binds by address to the mailbox you specify in the **LogonSMTP** parameters:

```
objSession.LogonSMTP("My Name", "myaddress@mycompany.com")
```

The identity of a user logging on with this syntax is not authenticated. The user can assume any arbitrary identity.

When CDO for NTS is running with IIS, the Inbox is a common folder shared by all recipients and applications, and containing all undeleted messages received by Microsoft® Internet Information Server (IIS). However, your application can access only those messages destined for the mailbox you bind to in the **LogonSMTP** call.

When CDO for NTS is running with the Microsoft Exchange Server, the Inbox is the regular Inbox of the messaging user's mailbox. When CDO for NTS is running with Microsoft MCIS 2.0 Mail, the Inbox is the messaging user's POP3 server Inbox.

The session is terminated by the **Logoff** method.

MessageFormat Property (CDONTS Session Object) Group

The **MessageFormat** property returns or sets the default message encoding. Read/write.

Syntax

objSession.**MessageFormat**

Data Type

Long

Remarks

The **MessageFormat** property has the following defined values:

| MessageFormat setting | Value | Description |
|-----------------------|-------|---|
| CdoMime | 0 | The message is in MIME format. |
| CdoText | 1 | The message is in uninterrupted plain text (default value). |

The **MessageFormat** property defaults to **CdoMime**. It serves as the default value for the **MessageFormat** property of the Message object.

Name Property (CDONTS Session Object)

Group

The **Name** property returns the display name used to log on to this session. Read-only.

Syntax

objSession.**Name**

The **Name** property is the default property of a Session object, meaning that *objSession* is syntactically equivalent to *objSession*.**Name** in Microsoft® Visual Basic® code.

Data Type

String

Example

```
' 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
```

Outbox Property (CDONTS Session Object)

Group

The **Outbox** property returns a Folder object representing the current messaging user's Outbox folder. Read-only.

Syntax

objSession.**Outbox**

Data Type

Object (Folder)

SetLocaleIDs Method (CDONTS Session Object)

Group

The **SetLocaleIDs** method sets identifiers that define a messaging user's locale.

Syntax

objSession.**SetLocaleIDs**(*CodePageID*)

objSession

Required. The Session object.

CodePageID

Required. Long. The code page identifier to be used for this messaging user.

Remarks

A locale is the set of features of a messaging user's environment that are dependent on language, country, culture, and conventions. These features include the character selection, the collating sequence and sort order, and the date, time, and currency formats. The **SetLocaleIDs** method sets identifiers that determine the behavior of locale-sensitive operations.

A code page identifier is a long integer specifying the ordered character set to use when displaying text. Information about a code page can be obtained from the Windows NT **GetCPInfo** function.

Before **SetLocaleIDs** is called, the locale and code page are set to the values in the user's registry.

If **SetLocaleIDs** is to be called, it must be called before the Session object's **LogonSMTP** method is called. This allows the messaging user's profile to be set for the appropriate locale. A call to **SetLocaleIDs** following logon returns **CdoE_CALL_FAILED**.

Note that the **SetLocaleIDs** method is the sole exception to the rule that a call to a session's **LogonSMTP** method must precede any other access to that session.

Every new message you create within the current session uses the locale established by your most recent call to the Session object's **SetLocaleIDs** method, or the default settings if you have never called **SetLocaleIDs**. A Message object created by the Messages collection's **Add** method cannot subsequently change its locale, even if **SetLocaleIDs** is called again. However, you can change a NewMail object's locale after its creation by calling the NewMail object's **SetLocaleIDs** method. Such a call does not affect the locale of any other Message or NewMail object, nor does it alter the settings established by the Session object's **SetLocaleIDs** method.

SetLocaleIDs tests the validity of the code page specified by the *CodePageID* parameter before actually setting the locale identifiers. If the code page is not valid, **CdoE_INVALID_PARAMETER** is returned.

Version Property (CDONTS Session Object)

Group

The **Version** property returns the version of the CDO for NTS Library. Read-only.

Syntax

objSession.**Version**

Data Type

String

Remarks

The **Version** property currently returns "1.2.1".

Samples

Microsoft provides several sample applications and sample wizards that illustrate the use of Microsoft® Collaboration Data Objects (CDO). In general, each of these samples performs a discrete messaging function, such as letting a user log on to access a Microsoft® Exchange [mailbox](#).

For information on obtaining and installing the sample applications, see [Installation and Use](#).

To use these samples, you need Microsoft Exchange Server installed on your system. Because these samples make use of the [CDO libraries](#), the Active Server™ components of Microsoft Exchange must also be installed. (Install them by selecting the **Outlook Web Access** option when installing Microsoft Exchange Server.)

The samples described in this section are shown in the following list. Within the first group (Sample Applications), the first ones perform the basic types of logging on and are followed by a simple discussion forum application. Then several applications cover nuts-and-bolts tasks such as sending e-mail. Finally, the pieces are put together in useful and illustrative applications such as Windows CE Mail, an online organization chart, and two applications based on the Discussion Forum – Culinary Corner (for restaurant reviews), and Classified Ads. The second group lists the sample wizards that ship with Microsoft Exchange Server version 5.5 or later.

Sample Applications

- [Authenticated Logon](#)
- [Anonymous Logon](#)
- [Discussion Forum](#)
- [Full Send](#)
- [Find User](#)
- [Find Items](#)
- [CE Mail](#)
- [Organization Chart](#)
- [Culinary Corner](#)
- [Classified Ads](#)
- [Job Candidates](#)
- [AutoCategory](#)
- [AutoAccept](#)
- [BankPost](#)

Sample Wizards

- [Microsoft Exchange Discussion Wizard for FrontPage](#)
- [Microsoft Exchange Form Design Wizard Sample](#)

Installation and Use

These samples are available from the Microsoft Exchange Application Farm on the World Wide Web. This site will also contain any upgrades to these samples and any new samples as they are written.

Group

To install a sample from the Web

1. Connect to the Microsoft Exchange Application Farm site at <http://www.microsoft.com/technet/appfarm>.
2. Under **Active Server Applications**, click an application to download, such as **Authenticated Logon**.
3. Click the hyperlink labeled **Click here to download ...**
4. In the **Internet Explorer** dialog box that warns about viruses, click **Save it to disk**.
5. In the **Save As** dialog box, move to a folder into which you want to download the sample application, and click **Save**. (If the desired folder does not exist, you can start Microsoft® Windows NT® Explorer and create the folder now.)
6. Using Microsoft Windows NT Explorer, find the folder where the application was saved. Double-click the .exe file in this folder (such as **authenticated.exe**). This .exe file self-extracts, populating the folder with the application's files.
7. Now, follow the installation instructions for the particular sample application (see the following section, **Installation Instructions: Compact Disk**). These instructions describe creating a folder (and possibly, subfolders) on your Microsoft® Internet Information Server (IIS) computer, making the top folder a virtual root, and changing its permissions to make the application available through a URL.

After you have completed these steps, you and others can run the application using a Web browser such as Microsoft® Internet Explorer.

Installation Instructions: Compact Disk

You can also install some of these sample applications and wizards from distribution compact disks. The description of each sample contains a section detailing specific compact disk installation instructions. These sections also provide information on preparing your computer before installation.

- [Installing Authenticated Logon](#)
- [Installing Anonymous Logon](#)
- [Installing Discussion Forum](#)
- [Installing Full Send](#)
- [Installing Find User](#)
- [Installing Find Items](#)
- [Installing CE Mail](#)
- [Installing Organization Chart](#)
- [Installing Culinary Corner](#)
- [Installing Classified Ads](#)
- [Installing Job Candidates](#)
- [Installing the Microsoft Exchange Discussion Wizard](#)
- [Installing the Form Design Wizard Sample](#)

Authenticating Users

Several of these samples require that you specify a Microsoft Exchange server name and a mailbox name. Use your assigned mailbox name (for example, "johnd") instead of your full name ("John Doe") for the mailbox name.

Also, you will often be presented with an **Authentication** dialog box, which requires valid credentials for the mailbox you are trying to access. Enter your Windows NT account name, using a backslash, as <Windows NT domain>\<user account name>. For example, if your domain name is southwest and your Windows NT account name is johnd, enter **southwest\johnd**.

Installed Directories and Files

After you have completed the application setup, application files reside in a directory tree starting at an IIS virtual root. Within this tree, each application consists of some combination of the following file types:

- **.asp files** Contain Active Server Pages scripts. These files can also include HTML code.
- **.inc files** Contain functions to be included in one or more ASP files.
- **.htm files** Traditional HTML files that do not contain Active Server Pages script.
- **.gif files** Graphic image files in GIF format used to depict items such as screen titles, buttons, and icons.

About Common Scripts

This section describes selected Microsoft® Active Server Pages (ASP) script files that are used by more than one CDO sample application. You can find these files in the subdirectories of the individual sample applications.

Some of these files are identical for different samples, but in most cases there are different variations that perform slightly different functions. For example, the global.asa file in the [Full Send](#) sample application provides functionality different from that of the global.asa file in the [Anonymous Logon](#) sample application.

About global.asa

Each of these sample applications uses a script called global.asa. This file contains Microsoft® Visual Basic® Scripting Edition (VBScript) code, it is run on the server, and its functions are called by Active Server Pages. The two main functions of global.asa create the necessary Application and Session objects for a session that is just beginning, make system services available, and handle impersonation to ensure that the Microsoft® Internet Information Server (IIS) thread for a given user is associated with the correct logon credentials.

Global.asa is the starting point for every ASP application. Typically, global.asa contains the following four VBScript functions:

- **Application_OnStart**
- **Application_OnEnd**
- **Session_OnStart**
- **Session_OnEnd**

For more information about global.asa, see the topic "Using the Global.asa File" in the Active Server Pages Roadmap. You receive the Roadmap when you install IIS.

The `Application_OnStart` Function

The `Application_OnStart` function is called before any .asp files are processed – before any text or graphics are rendered and sent to the user's browser. Within this function, the following call to the `CreateObject` method on the Active Server Pages Server object creates the CDO Rendering Library `RenderingApplication` object. If this call succeeds, the `objRenderApp` variable contains a pointer to the new object.

```
Set objRenderApp = Server.CreateObject("AMHTML.Application")
```

With the pointer retrieved, it is now possible to work with this CDO Rendering Library object – to call its methods and set its properties. Because a single `RenderingApplication` object can be shared by all currently active sessions, this is the only time the object needs to be created. The pointer to the `RenderingApplication` object is stored in the Active Server Pages Application object, using the following call.

```
Set Application("RenderingApplication") = objRenderApp
```

This call makes this instance of the object globally accessible so that any session can use it. The "RenderingApplication" tag is assigned arbitrarily and used to identify the storage location of the pointer to the CDO Rendering Library object. You can later use this tag to retrieve the pointer because it differentiates the object from anything else stored in the Active Server Pages Application object. Now, during processing of any ASP script in this virtual root, the `RenderingApplication` object can be retrieved from the Active Server Pages Application object.

Next, the impersonation handle on the Active Server Pages Application object is initialized with the following call. Strictly speaking, this call is unnecessary because this handle is empty before it is set. However, this call is used to remind the developer that the `hImp` variable needs to be reset later, in the `Application_OnEnd` function.

```
Application("hImp") = Empty
```

The Application_OnEnd Function

The **Application_OnEnd** function is called when the application ends, which occurs when IIS shuts down. It removes the Application object from memory. The Application object is not persistent, meaning that any data stored in the Application object is lost when IIS shuts down.

```
Set Application("RenderingApplication") = Nothing
```

This function is not called when an individual user session ends; that is, it is unaffected by events such as time-outs and user-requested exits.

The Session_OnStart Function

The **Session_OnStart** function is called by Active Server Pages the first time a user request for this application arrives from a browser.

This function initializes the impersonation handle, the handle to the security context.

```
Session("hImp") = Empty
```

The impersonation handle on the Session object is initialized with the preceding call. Strictly speaking, this call is unnecessary because this handle is empty before it is set. This call is used to remind the developer that the *hImp* variable needs to be reset later in the **Session_OnEnd** function.

```
Set Session("AMSession") = Nothing
```

If logon.inc is later used, this cannot be set to **Empty**; rather, set it to **Nothing**. For more information, see the Active Server Pages and VBScript documentation.

The Session_OnEnd Function

The **Session_OnEnd** function cleans up objects when the user session ends, whether because of a time-out or a user exit, or when Active Server Pages abandons a session.

Both the **Session_OnStart** and **Session_OnEnd** functions are in response to specific user requests coming in from a browser. The first time a new user requests an .asp file, **Session_OnStart** is called. When **Session_OnStart** is finished, the application's first .asp file starts to process. **Session_OnEnd** is called either when the session times out or when there is a script call that tells Active Server Pages to abandon the session, such as when a user clicks **Log Off** in [Microsoft Outlook Web Access](#).

If you rewrite the **Session_OnEnd** function, make sure you return all session variables to **Empty** or to **Nothing** (for object variables).

Using the Impersonate Method

Impersonation is used in the **Session_OnEnd** function to ensure that a CDO object is in the right security context when the user session ends.

At logon time, the **ImpID** property is read. This property returns an impersonation handle that can be subsequently used in **Session_OnEnd**. The handle is stored in the Session object:

```
Session("hImp") = objRenderApp.ImpID
```

In the **Session_OnEnd** function the **Impersonate** method is used to impersonate a user context.

```
hImp = Session("hImp")  
objRenderApp.Impersonate(hImp)
```

The call

```
objRenderApp.Impersonate(0)
```

reverts the session to its initial state, that of an empty session. This must occur before the session terminates.

The Application object may have session handles that refer to stored objects (such as a folder, a message, or a renderer). In this case, you need to store an impersonation handle in the Application object so that the application is in the right security context when the application ends in **Application_OnEnd**.

This applies to the Session object as well; if you store any data in the Session object, you must also store an impersonation handle in the Session object so that you can destroy the right objects in the right security context in **Session_OnEnd**.

About logon.inc

The logon.inc script file creates and checks a CDO session through use of the following VBScript functions.

AuthenticateUser

The **AuthenticateUser** function ensures that the user is already authenticated. Note that this implies that Basic Authentication or NTLM Authentication is enabled on the IIS server. **AuthenticateUser** determines this by checking whether the AUTH_TYPE server variable of the Active Server Pages Request object contains the string "basic" or the string "NTLM".

This function should be called at the beginning of an .asp file, before any command that may cause text to be sent to the browser.

The following cases cause text to be sent to the browser and will cause the **AuthenticateUser** function to fail if they occur before the **AuthenticateUser** function call:

- HTML is sent on a page before user authentication is attempted.
- HTML comments (other than **<!--#include** statements) are encountered.
- **Response.Write** is called.

If authentication fails, the following commands are executed:

```
'Hold information coming from the server and do not send it to the browser until after  
Response.End  
Response.Buffer = TRUE
```

```
'Say "Access denied"  
Response.Status = ("401 Unauthorized")
```

```
'Failed to authenticate user. Send responses and stop executing the script  
Response.End
```

This file sets the access status with the **Response.Status** command, so that if the user is not already authenticated, this function returns error 401 (Access denied) and prompts the user for credentials.

If the user is not authenticated, script execution stops and the browser request is not fulfilled. The user is notified that access is denied in the current unauthenticated state.

Active Server Pages and IIS handle the next attempt at authentication and impersonation, if necessary. After a user is authenticated, this script runs in the user's security context. That is, it runs on the server as the user. This means that the script has access to exactly those network resources that the user can access.

CheckAMSession

The **CheckAMSession** function checks for a valid CDO session (a connection to the server), and returns the CDO session in the Session object. It should be at the beginning of the script. It is important to call **CheckAMSession** before sending any HTML to the browser; otherwise, any redirects, authentication, and so on, may not work.

If no session is found, **CheckAMSession** calls the **NoSession** function. If the CDO session does not exist and the URL does not contain server and mailbox data, a logon form will be sent to the user to fill out.

The **CheckAMSession** function returns **True** if a session exists or can be created.

The following call to the Session object retrieves the "AMSession" named property.

```
Set amSession= Session( "AMSession")
```

Previously, in the global.asa file, Session("AMSession") was set to **Nothing**. Now, in the first call to the **CheckAMSession** function, it still is **Nothing**. When it is **Nothing**, the **NoSession** function is called to create a session. Upon return, the existence of a session is checked once more, again in the call

```
Set amSession= Session( "AMSession")
```

After this, **CheckAMSession** is set to **True** if a session has been successfully recognized or started.

Other Functions in logon.inc

NoSession

The **NoSession** function is called when the CDO session cannot be found. It either creates a session or gets more information from the user. It returns only if the session was created.

In **NoSession**, first the **GetConfig** function is called to find the Microsoft® Exchange server and a Microsoft Exchange mailbox that respond to the valid Microsoft® Windows NT® account.

ReportError

The **ReportError** function prepares an error text string and sends it to the browser.

GetConfig

The **GetConfig** function finds a server and mailbox name to log in with. If necessary, this function returns a form to the user requesting the information.

GetConfig uses a query string, which contains parameters on the URL that will be used by **CheckAMSession** within logon.inc. An example query string is the text that follows the question mark in the following expression:

```
http://.../logon.asp?server=server&mailbox=mailbox
```

If a query string is present in the URL, the two values within it are returned. If one of the values is blank, HTML text is sent to the user requesting either of both of them.

Tip By replacing this subroutine, you can use hard-coded data or retrieve the server and mailbox names from another source, such as a database.

About anlogon.inc

The file anlogon.inc creates and checks a CDO session using the following VBScript functions.

CheckAMAnonFolders

The **CheckAMAnonFolders** function checks that the list of accessible folders (folders set by the owner to have a role for anonymous access) has been retrieved from the Microsoft Exchange directory. It does so with the following call:

```
Set amAnonFolders= Application( "AMAnonFolders")
```

If the list does not exist, it is retrieved with a call to the **NoAnonFolders** function. The **CheckAMAnonFolders** function then returns **True** if the folder list exists or has been created.

NoAnonSession

The **NoAnonSession** function is called (for example, by **CheckAMAnonFolders**) when an anonymous session cannot be found for the current user.

This function either creates a session through the following call:

```
Set objAMSession1 = Server.CreateObject("MAPI.Session")
```

or it calls **GetAnonConfig** to retrieve more information about the Microsoft Exchange server. After it has this information, it attempts to log on with this call:

```
objAMSession1.Logon "", "", False, True, 0, True, bstrProfileInfo
```

This function returns only if the session was created.

GetAnonConfig

The **GetAnonConfig** function gets configuration information needed to open an anonymous CDO session by retrieving this information from the Windows NT registry, using the following command:

```
objRenderApp.LoadConfiguration 1, _  
    "HKEY_LOCAL_MACHINE\System\CurrentControlSet\Services\MSExchangeWeb\  
Parameters"
```

This function could be rewritten to use hard-coded information, or it could retrieve configuration information using **Request.QueryString**.

Other Functions in anlogon.inc

CheckAMAnonSession

The **CheckAMAnonSession** function checks that an anonymous CDO session exists in the Application object, with the following function call:

```
Set amSession= Application( "AMAnonSession")
```

If the session does not exist, it creates one by calling the **NoAnonSession** function.

NoAnonFolders

The **NoAnonFolders** function builds a list of folders published by the Microsoft Exchange administrator. This list is stored as the **Application("AMAnonFolders")** variable and is suitable to use as a ContainerRenderer datasource.

priv_ReportError

The **priv_ReportError** function prepares an error text string and sends it to the browser.

Authenticated Logon Sample Application

This application is the minimal script needed to perform an authenticated logon and display an Inbox message list. Its file logon.asp performs an authenticated logon, the file logoff.asp logs the user off, and the file folder.asp displays selected contents of the Inbox folder.

The user must supply Microsoft® Windows NT® domain credentials, but you can open any mailbox on the Microsoft® Exchange server on which the Authenticated Logon sample application is installed, or on the organization this server is a part of. The mailbox must correspond to a valid Windows NT account on the network.

Installing Authenticated Logon

Group

To install the Authenticated Logon sample application

1. Ensure that Microsoft® IIS version 3.0 or later, Microsoft® Active Server Pages, and Microsoft Outlook Web Access are installed on the same server. To install Web Access, run server setup on the distribution compact disk of Microsoft Exchange Server, select **Complete/Custom**, and select the **Outlook Web Access** option. This installs the CDO Library, the CDO Rendering Library, and Web Access .asp files.
2. Create a folder on the IIS computer – for example, c:\CDOSamples\auth\. Copy all files from the appropriate folder on the distribution compact disk into this folder.
3. Using the Internet Service Manager, create a new virtual root indicating that folder, and enable both Read and Execute permissions on it. For example, if you create a virtual root called \auth, the files in it will be available through a URL such as http://<server>/auth/default.htm.

At this point, users should be able to move to and run this sample application using the URL based on the virtual root you created.

Authenticated Logon Files

This application includes the following files:

| File | Purpose |
|-------------------|---|
| default.htm | Takes the user to logon.asp. |
| <u>global.asa</u> | Performs Active Server™ startup and shutdown functions, including impersonation to validate the user. |
| <u>logoff.asp</u> | This .asp file logs the user off. That is, it causes the Active Server session to be abandoned. As a side effect, the CDO session is also destroyed. |
| <u>logon.asp</u> | This .asp file demonstrates the functions in logon.inc. It performs an authenticated logon in that it causes a CDO session to be created (or finds an existing CDO session if one has already been created by this user) and reads properties from the session. |
| <u>logon.inc</u> | This include file contains functions written in Microsoft® Visual Basic® Scripting Edition (VBScript) used to log on to a Microsoft Exchange Server. These functions are used in logon.asp. |

When you click a link to the \auth virtual root, Active Server Pages calls the **Session_OnStart** (and **Application_OnStart**, if necessary) function in global.asa. The logon.asp script is then run; it includes the logon.inc file that contains actual user-authentication commands.

About logon.asp

The logon.asp script file calls the following functions:

BAuthenticateUser ensures that the user is authenticated. See the description in [logon.inc](#).

CheckAMSession checks for a CDO session in the Session object and returns it. See the description in logon.inc.

It then retrieves the CDO session with the following command:

```
Set objAMSession= Session( "AMSession")
```

This call ensures that the CDO session has not timed out, or if it has, re-establishes a session.

If no session can be retrieved, an error message is displayed.

Then, in an HTML section, it displays data from the session.

```
Inbox name : <b><%= objAMSession.Inbox.Name %></b><br>
```

```
Outbox name: <b><%= objAMSession.Outbox.Name %></b><br>
```

Finally, it lets the user log on or log off by clicking a link to logon.asp or logoff.asp:

```
<a href="logon.asp">Logon.asp</a><br>
```

```
<a href="logoff.asp">Logoff.asp</a><br>
```

About logoff.asp

The logoff.asp script file logs the CDO session off. It first sends a notification to this effect to the browser, and then calls

Session.Abandon

This call causes Active Server Pages to call **Session_OnEnd** in the global.asa file.

However, it would be possible to explicitly log off (terminate the MAPI session) within logoff.asp – not global.asa – by setting all CDO session variables containing objects to **Nothing** and emptying all other CDO session variables.

Then, this script lets the user log on again with a link to logon.asp:

```
<a href="logon.asp">Logon.asp</a>
```


Using Authenticated Logon

Group

To log on using the Authenticated Logon sample application

1. Start Authenticated Logon. A logon form is displayed by logon.asp.
2. Enter the name of your Microsoft Exchange server and the name of your mailbox on that server. Click **OK** or press **Enter**.
3. If you are not already logged on, the **Authentication** dialog box is displayed. In the **Username** text box, enter your Windows NT domain and account name separated with a backslash (no spaces), for example **southwest\johnd**. In the **Password** text box, type your Windows NT password. Click **OK** or press **Enter**.
4. Session data is now displayed, including the names of the Inbox and Outbox in your mailbox. You can now log on again (for example, to a different mailbox) by clicking a link to logon.asp, or log off by clicking a link to logoff.asp.

Anonymous Logon Sample Application

This is a simple application that logs on anonymously by creating an anonymous CDO session. It also finds and displays a list of published public folders. The anonfldr.asp file checks for and if necessary creates an anonymous CDO session on the Microsoft® Exchange Server. A list of messages in each published folder is displayed.

Installing Anonymous Logon

Group

To install the Anonymous Logon sample application

1. Ensure that Microsoft® IIS version 3.0 or later, Microsoft® Active Server Pages, and Microsoft Outlook Web Access are installed on the same server. To install Web Access, run server setup on the distribution compact disk of Microsoft Exchange Server, select **Complete/Custom**, and select the **Outlook Web Access** option. This installs the CDO Library, the CDO Rendering Library, and Web Access .asp files.
2. Create a folder on the IIS computer – for example, c:\CDOSamples\anon\. Copy all files from the appropriate folder on the distribution compact disk into this folder.
3. Using the Internet Service Manager, create a new virtual root indicating that folder, and enable both Read and Execute permissions on it. For example, if you create a virtual root called \anon, the files in it will be available through a URL such as http://<server>/anon/default.htm.
4. Make sure that the Anonymous user role has been given at least Reviewer permission within each public folder you will want to open. A folder owner can make this setting by first selecting the folder and then using the **Properties** option on the **File** menu of Microsoft® Outlook™. (By default, Anonymous users are given no access to public folders.) Additionally, if you intend to have a folder appear within Microsoft Outlook Web Access or if it is to be rendered through the RenderingApplication object, make sure the folder has been explicitly published. Microsoft Exchange administrators publish a folder using the **Folder Shortcuts** tab of the **HTTP (Web) Site Settings** property page in the <site name>\Configuration\Protocols container, using the Microsoft Exchange Administrator program.

At this point, users should be able to move to and run this sample application using the URL based on the virtual root you created.

Anonymous Logon Files

This application includes the following files:

| File | Purpose |
|---------------------|---|
| default.htm | Takes the user to anonfldr.asp. |
| <u>anlogon.inc</u> | Contains all of the Microsoft® Visual Basic® Scripting Edition (VBScript) code necessary to log on anonymously (create and check a CDO session), and find any published public folders. |
| <u>anonfldr.asp</u> | Checks for and if necessary creates an anonymous CDO session on the Microsoft Exchange Server, and then displays a list of messages in each published folder. |
| <u>global.asa</u> | Performs Active Server™ startup and shutdown functions, including impersonation to validate the user. |

About anonfldr.asp

Using the functions in [anlogon.inc](#), this file checks for and if necessary creates an anonymous CDO session on the Microsoft Exchange Server. It also renders one page of messages (25 lines) from each top-level published folder using the default view and all default renderer options.

First, it checks for the existence of an anonymous session by calling the **CheckAMAnonSession** function in anlogon.inc. If this session exists, the following command makes the anonymous session object available locally:

```
Set objAMAnonSession= Application( "AMAnonSession")
```

This file then checks for a list of published public folders by calling the **CheckAMAnonFolders** function in anlogon.inc. If this folder list is retrievable, the following command makes the folder list available locally:

```
amFolders= Application( "AMAnonFolders")
```

Next, it creates a ContainerRenderer object with this call:

```
Set objFolderRenderer = objRenderApp.CreateRenderer(3)
```

It then uses the ContainerRenderer object to render folders, using the amFolders list, with these calls:

```
'now render a table of the first 25 messages in the folder  
objFolderRenderer.DataSource = objFolder.Messages  
objFolderRenderer.Render 1,0,0,Response
```

Using Anonymous Logon

Group

To log on using the Anonymous Logon sample application

Click **Anonymous Logon**. You are logged on automatically. A one-page list of Microsoft Exchange Server public folders is rendered; you also see the messages they contain. These are the folders that an administrator has published using the **Folder Shortcuts** tab of the **HTTP (Web) Site Settings** property page in the **<site name>\Configuration\Protocols** container, using the Microsoft Exchange Administrator program. These folders are published on the server that is hosting the Anonymous Logon sample application.

Discussion Forum Sample Application

This is a Web-site-based discussion forum. Content is hosted in a Microsoft® Exchange public folder; the discussion is rendered under a single frame that appears as a frame on your Web site. This means you can use this sample as a template for a discussion forum on your own Web site with conversation threads stored in a Microsoft Exchange public folder.

This sample application uses the anonymous logon sample, and extends it in a practical way that demonstrates how to make use of discussion services of Microsoft Exchange Server for the benefit of http/browser users.

This sample application constitutes the basis for all the peer-to-peer discussion groups hosted on <http://www.exchangeserver.com>, as well as the Application Design Discussion on the Microsoft Exchange Application Farm (<http://www.microsoft.com/technet/appfarm>).

Installing Discussion Forum

Group

To install the Discussion Forum sample application

1. Ensure that Microsoft® IIS version 3.0 or later, Microsoft® Active Server Pages, and Microsoft Outlook Web Access are installed on the same server. To install Web Access, run server setup on the distribution compact disk of Microsoft Exchange Server, select **Complete/Custom**, and select the **Outlook Web Access** option. This installs the CDO Library, the CDO Rendering Library, and Web Access .asp files.
2. Create a folder on the IIS computer – for example, c:\CDOSamples\discussion\. Copy all files from the appropriate folder on the distribution compact disk into this folder.
3. Using the Internet Service Manager, create a new virtual root indicating that folder, and enable both Read and Execute permissions on it. For example, if you create a virtual root called \discussion, the files in it will be available with a URL like http://<server>/discussion/default.htm.
4. Select or create a Microsoft Exchange public folder to be used for the discussion. Make sure that the Anonymous user has at least Create Items and Read Items permissions on that folder.
5. Find the folder identifier of the Discussion Forum public folder. To do this, log on with Microsoft® Outlook™ Web Access; move to the public folder, and click the **Update Page Address** icon to update the URL in the browser's **Address** box. The address will appear like the following:
http://<server>/exchange/pf/root.asp?
obj=000000001A447390AA6611CD9BC800AA002FC45A0300746FC765786ED0119029
00AA00A71AE3000000000120000

6. Carefully copy this folder identifier number and paste it into the root.htm file after the word **folderID=**, as shown here:

```
<FRAME  
SRC="discroot.asp?  
folderID=000000001A447390AA6611CD9BC800AA002FC45A0300746FC765786ED0119  
02900AA00A71AE30000000069F00000" name="ExchangeDiscussion_fr"  
marginheight=0 marginwidth=0 >
```

There is an another way to obtain the identifier of a folder. If the folder that you are linking to is an immediate subfolder of **All Public Folders**, you may need to use the Alternative Method in the section described in the Classified Ads sample application.

Discussion Forum Files

This sample application includes the following files:

| File | Purpose |
|-----------------------------|--|
| amprops.inc | Defines common CDO constants and properties. |
| anlogon.inc | Contains all of the Microsoft® Visual Basic® Scripting Edition (VBScript) code necessary to log on anonymously (create and check a CDO session), and find any published public folders. |
| compose.asp | When the user clicks Post New Item or Reply , compose.asp displays a form (a new browser instance) to write a message in and then creates the message in the discussion folder. If the user is replying to a message, the form fields are preloaded with data from the original message. |
| discroot.asp | The root frame for the discussion. Called from root.htm, this file calls title.asp and folder.asp to display the discussion content. This file also contains several JavaScript functions called by the browser on events in title.asp and folder.asp. |
| existing.htm | This is a placeholder for your existing Web site. It is displayed at the top of your browser window. It contains links to two discussions. |
| folder.asp | Folder.asp calls functions in folder.inc to set the renderer and to render views of folders. It displays a page of messages (25 lines at a time). The format depends on the view selected. Each message provides a link to read.asp to allow that message to be read. |
| folder.inc | This include file contains most of the code specific to the CDO Rendering Library. Rendering objects are created and set up here; folder views, current folder, and the current page are set here. The folder.inc file contains functions for generating a renderer. |
| global.asa | Performs Active Server™ startup and shutdown functions, including impersonation to validate the user. |
| item.asp | This script processes next/previous message requests. |
| lang.inc | Sets strings and error messages. Concentrating these in a single include file eases localization. |
| read.asp | This script opens and displays the contents of a message (in a new browser instance) when you click a link to open the message. It contains controls for closing the message, going to the next or previous message, and replying to the message. |
| root.htm | This is a placeholder for your existing Web site. It shows how to link to discussion scripts and how to set them to open a specific folder. This file simulates existing content (discussion threads), and it creates a frameset for the menu (existing.htm) and diskroot.asp. |
| title.asp | This is a control bar containing JavaScript controls for setting the view, refreshing the screen, paging through |

messages, and launching `compose.asp` (through a link).

About folder.asp

The folder.asp file renders the message contents of a folder. This script file calls the following functions:

BAuthenticateUser is called to ensure that the user is authenticated. See the description in [logon.inc](#).

CheckAMSession is called to check for and return a CDO session in the Session object. See the description in logon.inc.

The folder.asp script retrieves the CDO session with the following command:

```
set objAMSession= Session( "AMSession")
```

If no session can be retrieved, an error message is displayed.

This file attempts to create a rendering object to render the Inbox folder of the logged-on user. In the following line, it opens the Inbox folder object for reading:

```
objFolderRenderer.DataSource= objAMSession.Inbox.Messages
```

The following line displays the user's Inbox folder and its contents:

```
objFolderRenderer.Render 1, 0, 0, Response
```

Using Discussion Forum

Group

To use the Discussion Forum sample application

1. Start the Discussion Forum sample application. The registered public folders are displayed. Hyperlinks to posted messages within folders are available. The property on which the hyperlink is rendered changes according to the view selected.
2. To open a message with an active link, click the message. The file read.asp is called when you do this.
3. To reply to this message, click **Reply**. This displays the compose form (compose.asp) in the same active window. When you reply to a message, you append your message to the existing thread, after the current message. When you post a message, you put a message into a new conversation thread.

Full Send Sample Application

This sample application lets you create a full-fledged e-mail message that includes Cc: and Bcc: fields. This sample application uses the [authenticated logon sample application](#). Then, after authentication, the script rootcomp.asp controls the basic functioning of the application, to let you send mail to any recipient.

Installing Full Send

Group

To install the Full Send sample application

1. Ensure that Microsoft® IIS version 3.0 or later, Microsoft® Active Server Pages, and Microsoft Outlook Web Access are installed on the same server. To install Web Access, run server setup on the distribution compact disk of Microsoft® Exchange Server, select **Custom Install**, and select the **Active Server Pages** option. This installs the CDO Library, the CDO Rendering Library, and Web Access .asp files.
2. Create a folder on the IIS computer – for example, c:\CDOSamples\fullsend\. Copy the files (rootcomp.asp, compose.asp, resolve.asp, global.asa, logon.inc and all .gif files are mandatory; default.htm is optional) from the appropriate folder on the distribution compact disk into this folder.
3. Using the Internet Service Manager, create a new virtual root indicating that folder, and enable both Read and Execute permissions on it. For example, if you create a virtual root called \fullsend, the files in it will be available through a URL such as http://<server>/fullsend/logon.asp. You can do this by following these steps:
 1. Open the Microsoft Internet Service Manager. Read the properties of the "WWW" service.
 2. Click the **Directories** tab.
 3. Click **Add**.
 4. Enter a path to the folder created in step 2.
 5. Enable both Read and Execute permissions.
 6. Add a virtual root name.

At this point, users should be able to move to and run this sample application using the URL based on the virtual root you created, for example, http://<server name>/<virtual root name>/rootcomp.asp.

Full Send Files

This sample application includes the following files:

| File | Purpose |
|-------------------|--|
| compose.asp | Compose message form. First, it authenticates the user (sets up a CDO session). Then it is used for input of the intended message recipients, the message subject, and the message body. Calls resolve.asp when the user submits the form. |
| default.htm | Takes the user to rootcomp.asp. |
| <u>global.asa</u> | Performs Active Server™ startup and shutdown functions, including impersonation to validate the user. |
| logon.inc | Include file containing functions that perform basic logon tasks. |
| resolve.asp | Resolves recipients. This file retrieves input from compose.asp and checks for valid recipients. After this is done, resolve.asp attempts to send the e-mail message. |
| rootcomp.asp | The root frame for the compose note. This file sets up a frameset for compose.asp and resolve.asp to display the compose-message form. This file also contains the JavaScript function checkInput , which verifies that at least one recipient name has been entered. |

Resolve.asp creates and adds a message object to the Outbox's Messages collection. It stores recipient data and message text retrieved from the form in rootcomp.asp and then resolves recipients. If resolution is successful, it sends the message with:

```
Message.Send( saveCopyFlag )
```

Using Full Send

Group

To run Full Send

1. Start Full Send. An HTML logon form is displayed by the file rootcomp.asp.
2. Enter the name of your Microsoft Exchange server and the name of your mailbox on that server. Click **OK** or press **Enter**.
3. A Compose Message form is displayed by compose.htm. Use this form frames to specify To, Cc, Bcc, message Subject, and the Message text. You must specify at least one recipient before you can send the message. Click the send icon to send the message, or **Close** to exit.

Find User Sample Application

Use this application to search for a recipient by e-mail name, and have the details displayed. This sample application uses the authenticated logon sample.

Installing Find User

Group

To install the Find User sample application

1. Ensure that Microsoft® IIS version 3.0 or later, Microsoft® Active Server Pages, and Microsoft Outlook Web Access are installed on the same server. To install Web Access, run server setup on the distribution compact disk of Microsoft® Exchange Server, select **Custom Install**, and select the **Active Server Pages** option. This installs the CDO Library, the CDO Rendering Library, and Web Access .asp files.
2. Create a folder on the IIS computer – for example, c:\CDOSamples\findusr\. Copy all files from the appropriate folder on the distribution compact disk into this folder.
3. Using the Internet Service Manager, create a new virtual root indicating that folder, and enable both Read and Execute permissions on it. For example, if you create a virtual root called \findusr, the files in it will be available through a URL such as http://<server>/findusr/default.htm.

Find User Files

This sample application includes the following files:

| File | Purpose |
|-------------------|--|
| amprops.inc | Defines common CDO and MAPI constants and properties. |
| default.htm | Takes the user to findusr.asp. |
| details.asp | This .asp file renders, in a new window, the details (full name, phone number, manager, office, and so on) of the user being sought. It uses the ObjectRenderer object to render these details. |
| findusr.asp | Displays a form to get the Microsoft Exchange server, the user's mailbox, and the e-mail name of the person being sought. This file uses the functions (such as GetConfig) in logon.inc. It obtains the global address book (GAL) from the CDO session and sets the filter for the e-mail name being sought. |
| <u>global.asa</u> | Performs Active Server™ startup and shutdown functions, including impersonation to validate the user. |
| lang.inc | Sets strings and error messages. Concentrating these in a single include file eases localization. |
| <u>logon.inc</u> | This include file contains functions written in Microsoft® Visual Basic® Scripting Edition (VBScript) used to log on to a Microsoft Exchange Server. These functions are used in findusr.asp. |

Using Find User

Group

To run Find User

1. Start Find User. A logon form is displayed by the file findusr.asp.
2. Enter the name of your Microsoft Exchange server, the name of your mailbox on that server, and the e-mail name (alias) of the person you are looking for. Click **OK** or press **Enter**. If the person is found, address book information about that person is displayed.
3. You can click the person's name to see additional information, which is displayed by details.asp in a new window. These items are displayed with a combination of HTML and VBScript, invoking CDO Rendering.

Find Items Sample Application

Use this sample application to search for messages in a folder (for example, your Inbox) based on fields such as From, Sent To, Subject, Message Body, Importance, Read Status, and Message Size. This sample application uses the authenticated logon sample. It is written in JavaScript, HTML, and ASP.

Installing Find Items

Group

To install the Find Items sample application

1. Ensure that Microsoft® IIS version 3.0 or later, Microsoft® Active Server Pages, and Microsoft Outlook Web Access are installed on the same server. To install Web Access, run server setup on the distribution compact disk of Microsoft® Exchange Server, select **Custom Install**, and select the **Active Server Pages** option. This installs the CDO Library, the CDO Rendering Library, and Web Access .asp files.
2. Create a folder on the IIS computer – for example, c:\CDOSamples\FindItems\. Copy all files from the appropriate folder on the distribution compact disk into this folder.
3. Using the Internet Service Manager, create a new virtual root indicating that folder, and enable both Read and Execute permissions on it. For example, if you create a virtual root called \FindItems, the files in it will be available through a URL such as http://<server>/FindItems/default.htm.

Find Items Files

This sample application includes the following files:

| File | Purpose |
|-------------------|---|
| amprops.inc | Defines common CDO constants and properties. |
| default.htm | Takes the user to findusr.asp. |
| fimid.asp | Search criteria frame. The form data is posted to fimsg.asp in the lower frame. |
| fimg.asp | Creates a message filter and displays a table listing the found messages. |
| fimgdef.asp | Default (blank) results frame loads first with instructions. |
| findmsg.asp | Frameset page that displays the search criteria and the results frames. |
| <u>global.asa</u> | Performs Active Server™ startup and shutdown functions, including impersonation to validate the user. |
| lang.inc | Sets strings and error messages. Concentrating these in a single include file eases localization. |
| <u>logon.inc</u> | This include file contains functions written in Microsoft® Visual Basic® Scripting Edition (VBScript) used to log on to a Microsoft Exchange server. These functions are used in logon.asp. |
| read.asp | Opens and displays a message chosen from the results list (clicked by the user). |

Using Find Items

Group

To run Find Items

1. Start Find Items. A logon form is displayed, as the file findmsg.asp sets up the screen and opens an initial CDO session.
2. Enter the name of your Microsoft Exchange server and the name of your mailbox on that server. Click **OK** or press **Enter**.
3. The **Authentication** dialog box is displayed. In the **Username** text box, enter your Microsoft® Windows NT® domain and account name separated with a backslash (no spaces), for example **southwest\johnd**. In the **Password** text box, type your Microsoft® Windows NT® password. Click **OK** or press **Enter**. The file fimid.asp displays the Find Items text boxes at the top of the screen, while fimgdef.asp displays directions below.
4. Enter information in one or more fields to restrict your search. When specifying a person in the From or Sent To field, use the person's whole name, not the person's e-mail name. Fill in at least one of the first four fields. Now click **Find**. Fimid.asp submits this information to fimg.asp, which renders messages in your Inbox that match the specified criteria; they are displayed in the lower half of your screen.
5. Clicking a displayed message link opens the message in a separate window. It is rendered by read.asp. The message is displayed in rich text, converted into HTML by the renderer.
6. Click **Close** to close the application window.

CE Mail Sample Application

This sample application is an HTML mail client. It uses CDO Rendering to render the contents table of the folder being displayed, as well as images for the parent and child folders of the current folder. Otherwise, it renders using HTML text instead of more complicated rendering (with, for example, JavaScript code).

Although this application was designed for a system with limited resources (for example, navigation is restricted to the user's own mailbox on the server) it is a full-featured mail client. Its scripts were written specifically with the Microsoft® Windows® CE device and Pocket Internet Explorer 1.1 in mind, but it will work on nearly any browser. In contrast to other scripts shipping with Microsoft® Exchange Server (version 5.0 or later), the CE Mail scripts do not use frames and do not use client-side scripts. They are well suited to low-bandwidth or low-end browser clients.

The basic functionality of this application builds from the [Authenticated Logon](#) sample application and several of the other samples available on <http://www.microsoft.com/technet/appfarm>.

Installing CE Mail

Group

To install the CE Mail sample application

1. Ensure that Microsoft® IIS version 3.0 or later, Microsoft® Active Server Pages, and Microsoft Outlook Web Access are installed on the same server. To install Web Access, run server setup on the distribution compact disk of Microsoft Exchange Server, select **Complete/Custom**, and select the **Outlook Web Access** option. This installs the CDO Library, the CDO Rendering Library, and Web Access .asp files.
2. Create a folder on the IIS computer – for example, c:\CDOSamples\CEMail\. Copy all files from the appropriate folder on the distribution compact disk into this folder, keeping the hierarchy of subfolders intact.
3. Using the Internet Service Manager, create a new virtual root indicating that folder, and enable both Read and Execute permissions on it. For example, if you create a virtual root called \CEMail, the files in it will be available through a URL such as http://<server>/CEMail/logon.asp.

At this point, users should be able to move to and run this sample application using the URL based on the virtual root you created.

CE Mail Files

This application includes the following files:

| File | Purpose |
|-------------------|---|
| amprops.inc | Sets constants. |
| constant.inc | Sets string constants. |
| default.htm | Takes the user to logon.asp. |
| <u>folder.asp</u> | Renders folder contents. |
| <u>global.asa</u> | Performs Active Server™ startup and shutdown functions, including impersonation to validate the user. |
| item.asp | Opens and renders a message, when clicked. |
| logon.asp | Logs on and authenticates a user. |
| <u>logon.inc</u> | This include file contains functions written in Microsoft® Visual Basic® Scripting Edition (VBScript) used to log on to a Microsoft Exchange Server. These functions are used in logon.asp. |
| pickfold.asp | Renders folders contained within Inbox or other selected folder. |
| pickview.asp | Lets the user choose a view. |
| render.inc | Contains functions for obtaining renderer objects. |

As distributed, this sample application supports only one message class (one type of form), and thus contains only one subfolder under the Forms subfolder. This is the IPM subfolder.

Files In the Forms\IPM Subfolder

This folder contains .asp files for rendering a message.

| File | Purpose |
|-------------|---|
| read.asp | Renders a message, using HTML text only. |
| compose.asp | Establishes a CDO session. Also, opens a new page when you click Compose , Reply , Reply All , or Forward . |
| delete.asp | Lets the user delete a message. |

Using CE Mail

Group

To log on using the CE Mail sample application

1. Start CE Mail. A logon form is displayed.
2. Enter the name of your Microsoft Exchange server and the name of your mailbox on that server. Click **OK** or press **Enter**.
3. The **Authentication** dialog box is displayed. In the **Username** text box, enter your Microsoft® Windows NT® domain and account name separated with a backslash (no spaces), for example **southwest\johnd**. In the **Password** text box, type your Windows NT password. Click **OK** or press **Enter**.
4. To open a folder other than Inbox, click **Pick Folder**; otherwise, click **Inbox**. If you click **Inbox**, the script `folder.asp` is run, which renders the contents of your Inbox on the server. If you click **Pick Folder**, the file `pickfold.asp` displays the folder tree starting at your server Inbox to let you open a different folder. `Pickview.asp` obtains the folder GUID for the chosen folder. Then, in a loop, it reads and displays the possible views from the `RenderingApplication` object. Clicking one of these folders causes its contents to be rendered.
5. While a folder is being displayed, you can select from among several folder views, with choices such as **Normal**, **Unread**, **Group by From**, and **Group by Subject**. These views are defaults; they are set automatically in Microsoft Exchange Server. They can be altered within a binary client such as Microsoft® Outlook™ and within Microsoft Outlook Web Access.
6. With a folder displayed, you can click to open an item in the folder. When you do so, the `render.inc` file in the virtual root finds the correct `read.asp` file in a subfolder to open the item. This design lets the user open a message or form of any type (class) by rendering it with the proper `asp` files for that class. These files reside in the specific form's subfolder under the `Forms\IPM` subfolder. (Each subfolder contains the files – including `read.asp` – for rendering a specific type of supported form or message.)

Organization Chart Sample Application

You can use this sample application to browse organization information obtained from the Microsoft® Exchange Global Address List (GAL). This sample uses CDO, ASP, Java, and a custom C++ MAPI control (delivered in orgchart.dll).

This sample pulls organization information (a person's manager and direct reports) from the Microsoft Exchange Global Address List. Users can search by the person's name or by e-mail name and the resulting organization chart is displayed in a Java tree-control or in an HTML table.

The Organization Chart sample application is unique in the following way: The other sample applications described in this section project typical "groupware" Microsoft® Outlook™/Microsoft Exchange applications out to the Web.

In contrast, the Organization Chart sample application approaches application development from the point of view of a Web page designer, and builds the application using the platform and services supplied by Microsoft Exchange. In this respect, the directory development and navigation model presented in Organization Chart is different from a typical "groupware" client user interface.

Several technologies were used to create this application:

- Active Server™ to process scripts and return data to the client
- CDO to obtain data from Microsoft Exchange
- A small C++ ActiveX® control to obtain additional data from Microsoft Exchange
- Java to provide a rich user interface

The C++ and Java components are supplied in both binary and source form.

ASP Scripts

The scripts are based on the [Anonymous Logon sample application](#) and the [Find User sample application](#).

ActiveX Control

CDO exposes a **Manager** property on an AddressEntry object, but it does not expose a corresponding Reports collection of AddressEntry objects. For this reason, it was necessary to write this ActiveX control. It demonstrates how to take a CDO object and recover an **IMAPIProp** pointer from the **IDispatch** pointer.

The control exposes just three properties: **DataSource**, **Count**, and **Item**:

- **DataSource**: Set this property to point to an AddressEntry object.
- **Count**: After setting the DataSource, this property returns the number of Reports found.
- **Item(n)**: This property returns the identifier of an AddressEntry object corresponding to one of the Reports. The parameter n refers to an individual Report.

The following example in Active Server syntax examines all of the direct reports of the currently logged-on user:

```
Set oAEReports= Server.CreateObject("OrgChart.Reports")
oAEReports.DataSource= oSession.CurrentUser
For idx= 1 To oAEReports.Count
Set oRep= oSession.GetAddressEntry( oAEReports.Item(idx) )
' do whatever you need to do with the oRep object
Next
```

Installing Organization Chart

Group

To install the Organization Chart sample application

1. Ensure that Microsoft® IIS version 3.0 or later, Microsoft® Active Server Pages, and Microsoft Outlook Web Access are installed on the same server. To install Web Access, run server setup on the distribution compact disk of Microsoft Exchange Server, select **Complete/Custom**, and select the **Outlook Web Access** option. This installs the CDO Library, the CDO Rendering Library, and Web Access .asp files.
2. Create a folder on the IIS computer – for example, c:\CDOSamples\OrgChart\. Copy all files from the appropriate folder on the distribution compact disk into this folder, keeping the hierarchy of subfolders intact.
3. Using the Internet Service Manager, create a new virtual root indicating the subfolder "scripts" (c:\CDOSamples\OrgChart\scripts), and enable both Read and Execute permissions on it. For example, if you create a virtual root called \OrgChart, the files in it will be available with a URL like http://<server>/OrgChart/jamorgchart.asp.
4. Register the orgchart.dll in the \bin directory:
c:\AMSamples\OrgChart\bin> regsvr32 orgchart.dll

Organization Chart Files

Files in \orgchart\bin

| File | Purpose |
|--------------|--|
| orgchart.dll | The DLL in which the C++ ActiveX control is implemented. This component retrieves information from the Microsoft Exchange GAL. |

Files in \orgchart\scripts

| File | Purpose |
|--------------------|---|
| amorgchart.asp | Displays organization-chart information from the GAL, rendered as HTML. |
| amprops.inc | Defines constants. |
| <u>anlogon.inc</u> | Contains all of the Microsoft® Visual Basic® Scripting Edition (VBScript) code necessary to log on anonymously (create and check a CDO session), and find any published public folders. |
| default.htm | Takes the user to jamorgchart.asp. |
| details.asp | Displays user account details. |
| <u>global.asa</u> | Performs Active Server startup and shutdown functions, including impersonation to validate the user. |
| jamorgchart.asp | Displays organization-chart information from the GAL through a Java tree control. |
| jnamesearch.asp | Searches for a person's name in the GAL (Java version). |
| lang.inc | Sets strings and error messages. Concentrating these in a single include file eases localization. |
| namesearch.asp | Searches for a person's name in the GAL (HTML version). |

Files in \orgchart\scripts\java

| File | Purpose |
|------------------|-------------------|
| JOrgChart.class | Java source files |
| PersonNode.class | Java source files |
| TreeLayout.class | Java source files |
| TreeNode.class | Java source files |

Files in \orgchart\source\cpp

These are the C++ source files for orgchart.dll.

Files in \orgchart\source\java

| File | Purpose |
|----------------|--|
| JOrgChart.java | A Java applet that provides the user interface: the tree structure, navigation through the list, and so on. This is the Java front-end to the Organization Chart sample application. |

Using Organization Chart

Group

To run Organization Chart

1. Start Organization Chart. You are logged on anonymously. The file jamorgchart.htm displays an HTML form requesting information about the employee you are seeking: either an e-mail name or the person's first or last name. On this screen, you can also choose between two different implementations of Organization Chart: Java or HTML.
2. Enter search information about an employee, and click **Search**. You can use partial search here. This means that you can search using initial letters of the person's first or last name. Partial search is not available when searching by e-mail name.
3. When the employee is found, information is extracted from that person's address entry in the Microsoft Exchange address book and displayed. This includes the person's e-mail name, full name, title, office number, and telephone number. This information appears in a box that is positioned in a tree structure along with other boxes showing that person's manager and direct reports.
4. To change the view to focus on the person's manager or a direct report, click the box displaying the other person's information.

Specifying a Different Address Book Server

When you first run the Organization Chart sample application, it obtains data from a Microsoft Exchange Global Address List (GAL) that is specified by settings in your computer's Microsoft® Windows NT® registry. By editing the script code for this sample application, you can make it use a GAL that is hosted on a different server. To do this, follow these steps:

Group

To specify a different address book server

1. Locate the function **GetAnonConfig** at the end of the file `anlogon.inc` of this sample application.
2. Within this function, comment out the following lines:

```
objRenderApp.LoadConfiguration 1, _ "HKEY_LOCAL_MACHINE\System\  
CurrentControlSet\Services\MSExchangeWeb\Parameters"
```

```
If Not ReportError( "RenderingApplication.LoadConfiguration from registry") Then  
    bstrEnterprise= objRenderApp.ConfigParameter("Enterprise")  
    bstrSite      = objRenderApp.ConfigParameter("Site")  
    bstrServer    = objRenderApp.ConfigParameter("Server")  
End If
```

3. Add assignment statements in which you set the following string variables to the enterprise, site, and server whose GAL you want this application to use. For example:

```
bstrEnterprise = "Your_Enterprise"  
bstrSite      = "Your_Site"  
bstrServer    = "Your_Server"
```

After you change these settings, the Organization Chart sample application will obtain its data from the GAL located on the server called "Your_Server" in the site "Your_Site" within the organization called "Your_Enterprise".

Culinary Corner Sample Application

This sample application is intended to let users compose and respond to restaurant critiques. It shows how to create a discussion on your Web site with conversation threads stored in a Microsoft® Exchange public folder. You can provide a link to this discussion, which will then open within a frame on a multiframe page. This multiframe page can be your existing Web site. The file root.htm simulates this case with sample restaurant critiques. This sample application is built on the discussion forum sample application.

This sample can be best utilized in conjunction with the Culinary Corner Microsoft Exchange public folder application available on <http://www.microsoft.com/technet/appfarm>. That sample application includes a PST (Personal Folder Store) file which you use in creating the Culinary Corner public folder on your Microsoft Exchange server. Among other things such as data (messages), the .PST file provides the views that the .asp version will read and apply.

Installing Culinary Corner

Group

To install the Culinary Corner sample application

1. Ensure that Microsoft® IIS version 3.0 or later, Microsoft® Active Server Pages, and Microsoft Outlook Web Access are installed on the same server. To install Web Access, run server setup on the distribution compact disk of Microsoft Exchange Server, select **Complete/Custom**, and select the **Outlook Web Access** option. This installs the CDO Library, the CDO Rendering Library, and Web Access .asp files.
2. Create a folder on the IIS computer – for example, c:\CDOSamples\eatery\. Copy all files from the appropriate folder on the distribution compact disk into this folder, keeping the hierarchy of subfolders intact.
3. Using the Internet Service Manager, create a new virtual root indicating that folder, and enable both Read and Execute permissions on it.
4. Select or create the Microsoft Exchange public folder to be used for Culinary Corner. Ensure that any custom folder views and forms created in Microsoft® Visual Basic® have been properly installed.
5. Find the folder identifier of the Culinary Corner public folder. To do this, log on with Microsoft® Outlook™ Web Access; move to the public folder and click the **Update Page Address** icon to update the URL in the browser's **Address** box. The address will appear like the following:
http://<server>/exchange/pf/root.asp?
obj=000000001A447390AA6611CD9BC800AA002FC45A0300746FC765786ED011902900AA00A71AE30000000000120000

6. Carefully copy this folder identifier number and paste it into the root.htm file after the word **folderID=**, as shown here:

```
<FRAME  
SRC="discroot.asp?  
folderID=000000001A447390AA6611CD9BC800AA002FC45A0300746FC765786ED0119  
02900AA00A71AE30000000069F00000" name="ExchangeDiscussion_fr"  
marginheight=0 marginwidth=0 >
```

There is another way to obtain the identifier of a folder. If the folder you are linking to is an immediate subfolder of **All Public Folders**, you may need to use the Alternative Method in the section described in the Classified Ads sample application.

7. Change the Anonymous user's permissions on the Culinary Corner public folder; follow these steps:
 1. Open the Microsoft Exchange Server Administrator program.
 2. Select the folder **Org Name\Folders\Public Folders**.
 3. Select your Culinary Corner public folder
 4. On the **File** menu, click **Properties**.
 5. Click **Client Permissions** in the General pane.
 6. Select the **Anonymous** user from the list and change its role to **Author**.
8. Add the Culinary Corner public folder to your HTTP folder shortcuts; follow these steps:
 1. Open the Microsoft Exchange Server Administrator program.
 2. Open the container **Org\Site\Configuration\Protocols\HTTP Site Settings**.
 3. Switch to the folder shortcuts pane and click **New**.
 4. Find your Culinary Corner public folder and add it to the list.

Culinary Corner Files

As shown in the following tables, the Culinary Corner application contains a set of critique files and a set of culinary files. The first set is used for composing new critiques on restaurants while the second set is used for responding to existing critiques. Each of these sets contains a rootcomp.asp, resolve.asp, compose.asp, and read.asp file.

The application's root folder holds the include files needed by both samples, and the base code for the discussion application.

Root Folder

| File | Purpose |
|--------------------|--|
| <u>anlogon.inc</u> | Contains all of the Microsoft® Visual Basic® Scripting Edition (VBScript) code necessary to log on anonymously (create and check a CDO session), and find any published public folders. |
| folder.inc | This include file contains most of the code specific to the CDO Rendering Library. Rendering objects are created and set up here; folder views, current folder, and the current page are set here. Folder.inc contains functions for generating a renderer. |
| discroot.asp | The root frame for the discussion. Called from root.htm, this file calls title.asp and folder.asp to display the discussion content. This file also contains several JavaScript functions called by the browser on events in title.asp and folder.asp |
| title.asp | This is a control bar containing JavaScript controls for setting views, refreshing the screen, paging through messages, and launching \culinary\compose.asp to post a new restaurant review. |
| folder.asp | Folder.asp calls functions in folder.inc to set the renderer and render views of folders. It displays a page of messages (25 lines at a time). The format depends on the view selected. Each message provides a link to read.asp to allow that message to be read. |
| item.asp | This script processes next/previous message requests. |
| <u>global.asa</u> | Performs Active Server™ startup and shutdown functions, including impersonation to validate the user. |
| amprops.inc | Definitions of common CDO constants and properties. |
| lang.inc | Sets strings and error messages. Concentrating these in a single include file eases localization. |

Culinary Folder

| File | Purpose |
|--------------|--|
| rootcomp.asp | Used for rendering the frameset of the culinary page. This page also contains JavaScript code used to validate the data in compose.asp before allowing resolve.asp to send the custom form/message. Interacts with resolve.asp and compose.asp to preserve data in the page. |
| compose.asp | Used when composing a new critique on a restaurant. Gathers data entered by the user. |
| resolve.asp | Used for sending the composed critique. It retrieves data |

from compose.asp and attempts to send mail to the Culinary Corner folder.

read.asp Used when reading a critique on a restaurant. Responsible for retrieving data from the message and rendering it. Allows users to reply to a critique (that is, to make a critique of a critique).

Critique Folder

| File | Purpose |
|--------------|--|
| rootcomp.asp | Used for rendering the frameset of the critique of the critique page. This page also contains JavaScript code used to validate the data in compose.asp before allowing resolve.asp to send the custom form/message. Interacts with resolve.asp and compose.asp to preserve data in the page. |
| compose.asp | Used when responding to a critique on a restaurant. Gathers data entered by the user. |
| resolve.asp | Used for sending the composed critique. It retrieves data from compose.asp and attempts to send mail to the culinary corner folder. |
| read.asp | Used when reading the critique of a critique. Responsible for retrieving data from the message and rendering it. Allows users to reply a critique of a critique generating another critique of a critique. |

Using Culinary Corner

This sample application functions in a way similar to the [Discussion Forum sample application](#). However, it uses a different set of forms tailored to writing and reading restaurant reviews.

Group

To use the Culinary Corner sample application

1. Start Culinary Corner. You are logged on anonymously. Root.htm creates a frameset that uses restrnt.htm on the top of your screen, title.htm across the center, and folder.htm in the bottom half. Folder.asp displays a list of items within the Culinary Corner public folder set up during installation. Title.htm lets you choose from among a number of views of restaurant listings (such as by Locale, by Value, and by Restaurant type), refresh the screen to check for newly posted reviews, page through additional messages, or post a new review.
2. To open a review with an active link, click it. Read.htm displays the critique in a new browser window.
3. Click **Close** to close this window, or **Critique The Cuisine** to respond to the original critique (using compose.htm and resolve.htm).

Classified Ads Sample Application

This sample application is a classified advertisement discussion forum. You can use it to search for, post, and reply to for-sale and item-wanted classified ads.

For more information, see the description in [Form Example: A Classified Ad](#).

Installing Classified Ads

Group

To install the Classified Ads sample application

1. Ensure that Microsoft® IIS version 3.0 or later, Microsoft® Active Server Pages, and Microsoft Outlook Web Access are installed on the same server. To install Web Access, run server setup on the distribution compact disk of Microsoft® Exchange Server, select **Custom Install**, and select the **Active Server Pages** option. This installs the CDO Library, the CDO Rendering Library, and Web Access .asp files.
2. Create a folder on the IIS computer – for example, c:\CDOSamples\classad\. Copy all files from the appropriate folder on the distribution compact disk into this folder, keeping the hierarchy of subfolders intact.
3. Using the Internet Service Manager, create a new virtual root indicating that folder, and enable both Read and Execute permissions on it.
4. Select or create the Microsoft Exchange public folder to be used for the Classified Ad. Ensure that any custom folder views and forms created in Microsoft® Visual Basic® have been properly installed.
5. Find the folder identifier of the Classified Ad public folder. To do this, log on with Microsoft® Outlook™ Web Access; move to the public folder and click the **Update Page Address** icon to update the URL in the browser's **Address** box. The address will appear like the following:
http://<server>/exchange/pf/root.asp?
obj=000000001A447390AA6611CD9BC800AA002FC45A0300746FC765786ED0119029
00AA00A71AE30000000000120000

6. Carefully copy this folder identifier number and paste it into the root.asp file after the word **folderID=**, as shown here:
<FRAME SRC="discroot.asp?
folderID=000000001A447390AA6611CD9BC800AA002FC45A0300746FC765786ED0119
02900AA00A71AE30000000069F00000" name="ExchangeDiscussion_fr"
marginheight=0 marginwidth=0 >

There is another way to obtain the identifier of a folder. If the folder you are linking to is an immediate subfolder of **All Public Folders**, you may need to use this Alternative Method.

7. Make the folder visible through the Global Address List. To do this, follow these steps:
 1. Open the Microsoft Exchange Server Administrator program.
 2. Click the folder **Org Name\Folders\Public Folders**.
 3. Click the folder you wish to add to the Address Book.
 4. On the **File** menu, click **Properties**.
 5. Click the **Advance** tab.
 6. Clear **Hide From Address Book**.

Folder Identifier: Alternative Method

Group

To obtain the identifier of a folder

1. Move into the public folders to the parent of the folder that you are looking for (so that the discussion folder is visible in the list of subfolders); examine the source HTML of the subfolders frame; and locate the identifier corresponding to the target folder. In the following sample of the HTML source, the identifier is shown in bold:

```
<tr bgcolor=FFFFFF valign="top" align="left">  
<td WIDTH=16>&nbsp;  </td><td><FONT COLOR=000000 SIZE=2>  
<IMG BORDER=0 hspace=0 SRC='folder.gif' ALT='Folder' WIDTH=16 HEIGHT=16>  
<A HREF='javascript:parent.SetNewFolderPick("  
000000001A447390AA6611CD9BC800AA002FC45A0300746FC765786ED01190  
2900AA00A71AE30000000069F00000")'>a folder</A>  
</td></tr>
```

2. Carefully copy this folder identifier number and paste it into the root.asp (or root.htm) file after the word **folderID=**, as shown here:

```
<FRAME SRC="discroot.asp?  
folderID=000000001A447390AA6611CD9BC800AA002FC45A0300746FC765786  
ED011902900AA00A71AE30000000069F00000" name="ExchangeDiscussion_fr"  
marginheight=0 marginwidth=0 >
```

Classified Ad Files

This sample contains two sets of files. The first set, in the folder NewClassifiedAd, stores the ASP files necessary to create and view a classified ad form. The second set, in the folder Purchoffr, stores the files needed to create and view purchase offers and responses.

The root folder holds the include files needed by both samples and the base code for the discussion application. Also see the section on the [Discussion Forum sample application](#) for descriptions of some of these files:

Root Folder (Discussion) Files

| File | Purpose |
|----------------------------|---|
| amprops.inc | Definitions of common CDO constants and properties. |
| default.htm | Takes the user to root.asp. |
| discroot.asp | The root frame for the discussion. Called from root.htm, this file calls title.asp and folder.asp to display the discussion content. This file also contains several JavaScript functions called by the browser on events in title.asp and folder.asp. |
| existing.htm | This is a placeholder for your existing Web site. It is displayed at the top of your browser window. It shows how to link to the discussion/classified ad scripts and how to set them to open a specific folder or message. |
| folder.asp | Folder.asp calls functions in folder.inc to set the renderer and render views of folders. It displays a page of messages (25 lines at a time). The format depends on the view selected. |
| folder.inc | This include file contains most of the code specific to the CDO Rendering Library. Rendering objects are created and set up here; folder views, current folder, and the current page are set here. Folder.inc contains functions for generating a renderer. |
| getrend.inc | Contains functions for opening information stores and messages. |
| global.asa | Performs Active Server™ startup and shutdown functions, including impersonation to validate the user. |
| item.asp | This script processes next/previous message requests. |
| lang.inc | Sets strings and error messages. Concentrating these in a single include file eases localization. |
| root.asp | Logs you onto the Microsoft Exchange site. Creates a frameset and launches existing.htm, folder.inc, and discroot.asp. |
| title.asp | This is a control bar containing JavaScript controls for setting views, refreshing the screen, paging through messages, and launching \newclassad\compose.asp to write a new ad. |

NewClassifiedAd Files

| File | Purpose |
|--------------|--|
| rootcomp.asp | Used for rendering the frameset of the new classified ad. This page also holds several JavaScript functions that |

| | |
|--------------------------------|--|
| | validate data in compose.asp before allowing resolve.asp to send the custom message or form. |
| compose.asp | Displays an input form that the client uses to place an ad to buy or to sell items. Data is passed to resolve.asp after it has been verified by rootcomp.asp |
| resolve.asp | Retrieves the data entered by the client in compose.asp and attempts to send mail to the public folder called Classified Ads. |
| category.asp
and subcat.asp | Used to display the many categories and subcategories of items to be sold or purchased. |
| read.asp | Displays the contents of a previously generated classified ad and lets users create responses known as purchase offers. |
| amprops.inc | Sets constants. |

Purchoffr Files

| File | Purpose |
|--------------|--|
| rootcomp.asp | Used for rendering the frameset of the purchase offer. This page also holds several JavaScript functions that validate data in compose.asp before allowing resolve.asp to send the custom message or form. |
| compose.asp | Displays an input form that the client uses to respond to an ad to buy or sell items. Data is passed to resolve.asp after it has been verified by rootcomp.asp. This form is not posted to the Classified Ad folder; rather, it is sent to the creator of the classified ad. |
| resolve.asp | Retrieves the data entered by the client in compose.asp and attempts to send mail to the individual who initially posted the classified ad. |
| read.asp | Displays the purchase offer generated by another user. Viewers of this form can respond to the offer using the standard reply note. |
| amprops.inc | Sets constants. |

Using Classified Ads

Group

To use the Classified Ads sample application

1. Start Classified Ads. A logon form is displayed by root.asp.
2. Enter as a mailbox name either **demo1** or **demo2** and click **OK** or press **Enter**.
3. If you are not already logged on, the **Authentication** dialog box is displayed. In the **Username** text box, enter your Microsoft® Windows NT® domain and account name separated with a backslash (no spaces), for example **southwest\johnd**. In the **Password** text box, type your Windows NT password. Click **OK** or press **Enter**.
4. As soon as you are authenticated, root.asp creates a frameset that uses existing.htm on the top of your screen, title.htm across the center, and folder.htm in the bottom half. Folder.asp displays a list of items within the Classified Ads public folder set up during installation. Title.htm lets you choose views of ads (All Items, All Items for Sale, All Items Wanted, and Ads by Category and Subcategory), refresh the screen to check for newly posted messages, page through additional folders and messages, or post a new item.
5. To open an ad with an active link, click it. Read.htm displays the ad in a new browser window.
6. Click **Close** to close this window, or **Respond to Ad** to respond (using compose.htm).

Job Candidates Sample Application

The Job Candidates sample application uses Active Server Pages (ASP) files to present a system for organizing candidates for employment.

Using this sample application, you can store information about a candidate such as personal contact information, job history, resume, and the title of the position the candidate is seeking. Using a browser, you can enter data about new candidates and update present candidates. If you have Microsoft® Outlook™ or another e-mail client installed, you can also schedule interviews for the candidate.

The fact that this is an ASP sample application means that all processing takes place on the server. This is beneficial in that the resulting pages are sent to the client as browser-independent HTML.

System Requirements

The following components must be installed on your system to run the Job Candidates sample application:

- Microsoft® Windows NT® Server version 4.0 or later
- Microsoft® Exchange Server version 5.0 or later
- Microsoft® Internet Information Server (IIS) version 3.0 or later
- Active Server Pages
- (optional) To send messages, an e-mail client such as Microsoft Outlook

Installing Job Candidates

Group

To install the Job Candidates sample application

1. Ensure that Microsoft® IIS version 3.0 or later, Microsoft® Active Server Pages, and Microsoft Outlook Web Access are installed on the same server. To install Web Access, run server setup on the distribution compact disk of Microsoft Exchange Server, select **Complete/Custom**, and select the **Outlook Web Access** option. This installs the CDO Library, the CDO Rendering Library, and Web Access .asp files.
2. Create a folder for this sample application on your IIS computer – for example, c:\CDOSamples\Job Candidates\. Copy all files from the appropriate folder on the distribution compact disk into this folder.
3. Using the Internet Service Manager, create a new virtual root indicating that folder, and enable both Read and Execute permissions on it.
4. Open Microsoft Outlook and click **Open Special Folder** on the **File** menu, and then click **Personal Folder**. This displays the **Connect to Personal Folders** dialog box.
5. Find the directory into which you copied the Job Candidates files and double-click JobCand.pst. This creates a personal folder store called Job Candidates, and adds it to your current profile.
6. Find the root folder in the newly created Job Candidates personal folder store and open it. Within this personal folder store is a folder called Job Candidates.
7. Drag this Job Candidates folder into the information store called All Public Folders.
8. Set Editor permission on this folder for any person who wishes to use this sample application. For more information, see Setting User Permissions.
9. Ensure that any custom folder views and forms created in Microsoft® Visual Basic® for this Job Candidates folder have been properly installed.
10. Make the folder visible through the Global Address List. To do this, follow these steps:
 1. Open the Microsoft Exchange Server Administrator program.
 2. Click the folder **Org Name\Folders\Public Folders**.
 3. Click the folder you wish to add to the Address Book.
 4. On the **File** menu, click **Properties**.
 5. Click the **Advance** tab.
 6. Clear **Hide From Address Book**.

You are now ready to use the Job Candidates ASP sample application.

Setting User Permissions

Those who use this sample application to view candidates' information or schedule interviews should have their permissions in the Job Candidates folder set to Editor.

Group

To set user permissions

1. Start the Microsoft Exchange Server Administrator program.
2. Within Public Folders, double-click the Job Candidates public folder in the right pane. This displays the property sheet for this folder.
3. On the **General** page, click **Client Permissions**.
4. Add any users (or distribution lists) who will use the Job Candidates sample application, and grant Editor permission to these users. (Start by clicking **Add**. See the online Help for more information on this procedure.)
5. Click **OK** to exit this page and click **OK** again to exit the property sheet.

Job Candidates Files

This application includes the following files:

| File | Description |
|--------------|---|
| abandon.asp | Abandons the Active Server™ session. |
| buttons.asp | Receives input from the Logoff and New Candidates buttons and redirects the user to the appropriate page. |
| default.htm | Takes the user to frmroot.asp. |
| folder.asp | Renders the list of job candidates. |
| frmroot.asp | Sets up the main window with the frameset of title.htm, abandon.asp, and graphic1.htm. |
| graphic1.htm | Contains a graphic. |
| graphic2.htm | Contains a graphic. |
| jobcand.pst | The public folder that contains the Microsoft Outlook version of this sample application. |
| logoff.asp | Cleans up variables and logs the user off. |
| logon.inc | Contains logon subroutines. |
| new.asp | Sets up the overview.asp form to accept a new candidate |
| overview.asp | Contains the Candidate Overview form and tabs. It can show information for a selected candidate or be displayed empty (by new.asp) to accept information about a new candidate. |
| pscont.inc | Contains constants used throughout the application. |
| read.inc | Contains a server-side Visual Basic Scripting Edition (VBScript) function. |
| resume.asp | Contains the Resume form and tabs. |
| saved.asp | This file is run when candidate information has been saved. It lets the user log off or return to the candidates list page. |
| schedule.asp | Lets the user schedule an interview for a selected candidate by sending a message to the person who entered the information or a different interviewer. |
| session.inc | Contains session-management, error-handling, and utility subroutines. |
| sesutil.inc | Contains several important subroutines used for manipulating candidate data. |
| submit.asp | Submits candidate information to the public folder. |
| title.htm | Contains the title of the Web page. |
| work.asp | Contains the Work Experience form and tabs. |

Using Job Candidates

To execute any of these procedures except logging off, start on the candidates list page of this application. To go to that page from other pages of the application, click **Return To Candidates List**.

Group

To create a new candidate

1. Click **New Candidate**. A Candidate Overview form is displayed.
2. Enter the new candidate's information into the fields in the Candidate Overview section of the overview.asp file. To enter information into different sections, first click the button of the section whose information you want to enter.
3. When finished entering information, click **Submit**.
4. To return to the Job Candidates folder, click **Return To Candidates List**.

Group

To update a candidate

1. To update a candidate, first click the message displayed for that candidate.
2. Update the fields in a section you choose by first clicking the button for that section.
3. When finished entering the updated information, click **Submit**.

Group

To schedule an interview

1. To schedule an interview for the candidate, click the message displayed for that candidate, and then click **Schedule Interview**.
2. Type the e-mail address of the recipient of the interview message in the **To** field. If you want other people to receive copies of this message, enter their addresses in the **CC** field.
3. Enter the message you wish to send, or keep the default message, which is displayed.
4. To send the message, click **Send**.

Group

To log off

To log off the ASP and MAPI sessions, click **LogOff** on any page except the **Schedule Interview** page.

AutoCategory Sample Application

AutoCategory is a sample application implemented in an event script that is triggered by the **Folder_OnMessageCreated** and **Message_OnChange** events. It searches for certain words in the message body of a new or changed message. If at least one of these words (known as "keywords") is found, AutoCategory writes the word into the keyword field of that message.

This is useful because in Microsoft® Outlook™ you can change your message view to display custom fields, including the keyword field. With this field displayed, you can, for example, sort the messages in the folder by keyword.

Because the script defines specific keywords, you can change them by editing the script. The original keywords are TEST, DEBUG, ERROR, and INVENTORY.

Installing the AutoCategory Agent

Before installing the AutoCategory agent, be sure you have the Microsoft® Exchange Event Service installed properly. For more information, see "At a Glance: Setting up the Event Service" in the *Microsoft Exchange Server Programmer's Reference* in the Microsoft® Platform SDK documentation.

Group

To install the AutoCategory agent in a folder

1. Start Microsoft Outlook and locate the folder whose events you want a script to respond to.
2. Right-click this folder, or click **Folder** on the **File** menu.
3. Click **Properties** for that folder.
4. Click the **Agents** tab to display the **Agents** property page. (This tab is not displayed unless the Microsoft Exchange Event Service is installed properly).
5. Click **New**. This displays the **New Agent** dialog box.
6. To enable monitoring of both new and changed messages, select the check boxes **A new item is posted in this folder** and **An item is changed in this folder**.
7. Click **Edit Script**.
8. Paste the AutoCategory script code, which you can find in the section [AutoCategory Files](#).
9. Close all dialog boxes by clicking **OK** in each. The AutoCategory agent is now installed.

Debugging AutoCategory

You can debug the AutoCategory sample application to ensure that it is installed properly, or to test changes you make to the AutoCategory script. The script typically informs you of the result of AutoCategory script execution by writing to one or both of two event logs. To debug AutoCategory by reading these logs, choose from the following methods:

- Inspect the Microsoft® Windows NT® Event Viewer on the server on which the Microsoft Exchange Event Service is running. The higher the logging level, the more information you see about script performance.
- View the log file that is displayed through Microsoft Outlook. To open this file, click **Properties** for the monitored folder, click the **Agents** tab, click **New** or **Edit**, and then click **Logs**. For more information about using this event log feature, see the topics "About Event Logs" and "New Agent Dialog Box" in the *Microsoft Exchange Programmer's Reference* in the Microsoft® Platform SDK documentation.

AutoCategory Files

This sample application includes only a single file, the AutoCategory script. This script, called AutoCat.txt, is provided here. You can use it in the installation of this sample application, described in [Installing the AutoCategory Agent](#):

```
<SCRIPT RunAt=Server Language=VBScript>

'THIS CODE AND INFORMATION IS PROVIDED "AS IS" WITHOUT
'WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED,
'INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTIES
'OF MERCHANTABILITY AND/OR FITNESS FOR A PARTICULAR
'PURPOSE

'Copyright (c) by Microsoft 1993-1998
'
'-----
'FILE DESCRIPTION: Exchange Server Event Script
'-----
' Auto Category Event Script
'
' 1) Keywords are searched for in the main body of text when a message is posted.
' 2) Keywords are searched for in the main body of text when a message is changed.
'     Note: A changed message will rewrite existing keywords (ie: if no keyword
'     matches are made in the changed message then the keyword field will now be
'     blank).
' 3) The keywords are hardcoded in this script as globals

Option Explicit

'-----
' Global Variables
'-----

Dim g_MatchCnt           'The number of matched keywords
Dim g_NumOfKeywords     'Total number of keywords
Dim g_bstrDebug         'DebugString
Dim g_boolSuccess       'Success Boolean

'-----
' SEARCH ARRAYS
'-----

Dim g_SearchArray
g_SearchArray = Array("TEST", "DEBUG", "MEMO", "INVENTORY", "NOTE")
Dim g_MatchedArray
g_MatchedArray = Array("", "", "", "", "")

'-----
' Event Handlers
'-----
```

```
' DESCRIPTION: This event is fired when a new message is added to the folder
Public Sub Folder_OnMessageCreated

    Call DebugAppend(vbCrLf&" - AUTOCAT Folder_OnMessageCreated - ",False)
    Call AutoCategoryExecute
    Call DebugAppend("Action Info: boolSuccess = " & g_boolSuccess & "    Matched
Keywords = " & g_MatchCnt & "    Number Keywords Searched =
"&cstr(g_NumOfKeywords+1),False)
    Script.Response = g_bstrDebug
```

end Sub

'-----

```
' DESCRIPTION: This event is fired when a message in the folder is changed
Public Sub Message_OnChange

    Call DebugAppend(vbCrLf&" - AUTOCAT Message_OnChange - ",False)
    Call AutoCategoryExecute
    Call DebugAppend("Action Info: boolSuccess = " & g_boolSuccess & "    Matched
Keywords = " & g_MatchCnt & "    Number Keywords Searched =
"&cstr(g_NumOfKeywords+1),False)
    Script.Response = g_bstrDebug
```

End Sub

'-----

```
' DESCRIPTION: This event is fired when a message is deleted from the folder
Public Sub Folder_OnMessageDeleted
    'Not Used
End Sub
```

```
' DESCRIPTION: This event is fired when the timer on the folder expires
Public Sub Folder_OnTimer
    'Not Used
End Sub
```

'-+-+-+-----
' PRIVATE FUNCTIONS/SUBS
'-+-+-+-----

'-----
' Name: AutoCategoryExecute
' Area: App
' Desc: Search and Replace Keywords
' Parm: None
'-----

Private Sub AutocategoryExecute

On Error Resume next

Dim omsg 'Message Object
Dim omsgfields 'Message Fields Collection Object
Dim oStores 'Stores Collection
Dim oTemp 'Temporary Object


```

Dim Item          'Object

Dim bstrTarget    'upper case body text
Dim result        'Variant search result
Dim x             'For next Variable

g_NumOfKeywords = ubound(g_SearchArray)
g_MatchCnt = 0
g_boolSuccess = False

'Use Session Object and Message ID to get incoming Message.
Set omsg = EventDetails.Session.GetMessage(EventDetails.MessageID,Null)
If err.number <> 0 then
    Call DebugAppend("EventDetails.Session.Getmessage",True)

Else
    '--I've got the message now, loop and test for all keywords
    bstrTarget = ucase(omsg.Text)
    if len(bstrTarget) = 0 then
        Call DebugAppend("Message body is Empty.",False)

    else
        'Search for the text body for keywords.
        for x = 0 to g_NumOfKeywords
            result = Instr(bstrTarget,g_SearchArray(x))
            if result <> 0 then
                g_MatchedArray(g_MatchCnt) = g_SearchArray(x)
                g_MatchCnt = g_MatchCnt + 1
            end if
        next

        'Reduce the size of the array otherwise field show lots of suffixed commas
        Redim Preserve g_MatchedArray(g_MatchCnt-1)

        'Get Keywords Field
        set oTemp = omsg.Fields.item("Keywords")
        If err.number = 0 then
            'field already exists replace it
            oTemp = g_MatchedArray
            omsg.Update
            If err.number <> 0 then
                Call DebugAppend("omsg.update",True)

            Else
                g_boolSuccess = True
            End If

        Else
            'Failure to locate keyword field means that I must add it
            err.clear
            omsg.fields.add "Keywords",vbArray, g_MatchedArray
            If err.number <> 0 then
                Call DebugAppend("omsg.Fields.Add",True)

            Else

```

```
        omsg.update
        If err.number <> 0 then
            Call DebugAppend("omsg.update",True)

        Else
            g_boolSuccess = True
        End if
    End If
End If
End If
End If
```

End Sub

```
-----
' Name: DebugAppend
' Area: Debug
' Desc: Simple Debugging Function
' Parm: String Text, Bool ErrorFlag
-----
```

Private Sub DebugAppend(bstrParm,boolErrChkFlag)

```
    if boolErrChkFlag = True then
        if err.number <> 0 then
            g_bstrDebug = g_bstrDebug & bstrParm & "Failed: " & cstr(err.number) &
err.description & vbCrLf
            err.clear
        end if

    else
        g_bstrDebug = g_bstrDebug & bstrParm & vbCrLf
    end if
```

End Sub

</SCRIPT>

Using AutoCategory

In the following assignment, the AutoCategory script sets four keywords:

```
searcharray = Array("TEST","DEBUG","ERROR", "INVENTORY")
```

You can run this sample application using these keywords, or you can edit the script and replace them with others of your choosing. Because "keyword" is a field visible through Microsoft Outlook, you can create a view that displays keywords for messages, within a given folder.

Group

To run AutoCategory

1. Send or post a message to the folder in which the AutoCategory agent is installed, or change (and save) a message within that folder. If the message body contains one or more of the four keywords, the message's Keywords field is updated.
2. To see the Keywords field, you need to modify your view to display keywords. To do this, first open the folder in which the AutoCategory agent is installed. Then, right-click the field bar and click **Field Chooser**. Now, click **All Document Fields**. Finally, drag the Keywords field onto the field bar.

At this point, this view will show the contents of the Keywords field of any message in this folder.

Note When the AutoCategory script finds a keyword in a message, it writes it into the keyword field of the message, automatically overwriting the previous contents of that field. For this reason, you cannot use more than one AutoCategory agent in a folder to search messages for different sets of keywords. In this case, whichever agent runs second would overwrite the output of the agent that ran first.

To avoid this problem, you could consider changing each of these agents to append – not overwrite – its output to the keyword field.

AutoAccept Sample Application

AutoAccept is a sample application implemented in an event script that is triggered by the **Folder_OnMessageCreated** event. You install this sample script in the Inbox folder of the e-mail account of a conference room. When a meeting request is received, the script checks the room's free/busy time to see if it is available at the requested time. If it is free, this script automatically books the meeting and sends a meeting-accepted response to the requester. If it is busy, it sends a meeting-declined response.

Installing the AutoAccept Agent

Group

To install the AutoAccept agent

1. Using the Microsoft® Exchange Server Administrator program, create a mailbox for the conference room on a Microsoft Exchange Server computer.
2. Using the Microsoft Exchange Server Administrator program, grant the conference room's mailbox the permission needed to install scripts—for example, Author permission. For more information, see "Granting Permissions" in the *Microsoft Exchange Server Programmer's Reference* in the Microsoft® Platform SDK documentation.
3. Create a Microsoft Exchange Server profile for the conference room. For more information, see the topic "Create a user profile" in the online Help for Microsoft® Outlook™.
4. Log on to Microsoft Outlook as the conference room. Do not log on as yourself. (The script schedules the room according to the free/busy times of the logged-on account.)
5. Using Microsoft Outlook, locate the Inbox of the conference room, which is the folder whose events you want the AutoAccept script to monitor.
6. Right-click the conference room's Inbox, or click it and then click **Folder** on the **File** menu.
7. Click **Properties** for that folder.
8. Click the **Agents** tab to display the **Agents** property page. (This tab is not displayed unless the Microsoft Exchange Event Service is installed properly).
9. Click **New**. This displays the **New Agent** dialog box.
10. To enable monitoring of new messages, select the check box **A new item is posted in this folder**.
11. Click **Edit Script**.
12. Paste the AutoAccept script code, which you can find in [AutoAccept Files](#).
13. Close all dialog boxes by clicking **OK** in each. The AutoAccept agent is now installed.

AutoAccept Files

This sample application includes only a single file, the AutoAccept script. This script, called Autoaccept.txt, is provided here. You can use it in the installation of this sample application, described in [Installing the AutoAccept Agent](#):

```
<SCRIPT RunAt=Server Language=VBScript>
```

```
'THIS CODE AND INFORMATION IS PROVIDED "AS IS" WITHOUT  
'WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED,  
'INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTIES  
'OF MERCHANTABILITY AND/OR FITNESS FOR A PARTICULAR  
'PURPOSE
```

```
'-----  
'  
' NAME: Autoaccept.Txt  
'  
' FILE DESCRIPTION: Auto-accept mtg request sample for Exchange Scripting Agent  
'  
' Copyright (c) Microsoft Corporation 1993-1998. All rights reserved.  
'  
'  
' NOTE: This example uses the Meeting Item Object. See MAPI SDK for the  
' details of the properties, constants and methods of this object.  
'  
' NOTE: This example also show how you might integrate Jscript with VBScript.  
'-----
```

Option Explicit

```
'-----  
' Global Variables  
'-----
```

```
Dim g_bstrDebug    'Debug String
```

```
'-----  
'   CONSTANTS  
'-----
```

```
Dim g_Const_CDOMeetingItem  
Dim g_Const_CDOMeetingRequest  
Dim g_Const_CDOMeetingResponse  
Dim g_Const_CDOResponseDeclined  
Dim g_Const_CDOResponseAccepted  
Dim g_Const_CDOResponseTentative  
Dim g_Const_CdoFree  
Dim g_Const_CdoBusy  
Dim g_Const_CdoTentative  
Dim g_Const_CdoOutOfOffice
```

```
g_Const_CDOMeetingItem = 27
g_Const_CDOMeetingRequest = 1
g_Const_CDOMeetingResponse = 2 'Note: Not used but given for reference
g_Const_CDOResponseDeclined = 3
g_Const_CDOResponseAccepted = 4
g_Const_CDOResponseTentative = 2
```

```
g_Const_CdoBusy = "2" 'Constants for BusyFree result we show
g_Const_CdoFree = "0" 'an example for using CdoFree
g_Const_CdoOutOfOffice = "3"
g_Const_CdoTentative = "1"
```

```
'-----
'                               EVENT HANDLERS
'-----
```

```
' DESCRIPTION: This event is fired when a new message is added to the folder
Public Sub Folder_OnMessageCreated
```

```
    Dim oStores 'Stores Object
    Dim oTemp 'Temporary Object
    Dim oMtg 'Meeting Object
    Dim oAppt 'Appointment Object
    Dim oMtgResp 'Meeting Response Object
    Dim oUser 'User Object
    Dim Item 'Item Object
```

```
    Dim Resp 'Response integer variable
    Dim Tstart 'Start Time
    Dim Tend 'End Time
```

```
    Dim x
```

```
    Set oUser = EventDetails.Session.CurrentUser
```

```
    Call DebugAppend("AUTOACCEPT - Folder_OnMessageCreated",False)
```

```
    'Get Incomming Meeting Msg
```

```
    Set oMtg = EventDetails.Session.GetMessage(EventDetails.MessageID, Null )
```

```
    If oMtg.Class <> g_Const_CDOMeetingItem Then
```

```
        Call DebugAppend("Message is not a meeting request or response.",False)
```

```
    Else
```

```
        'Ok, It's a meeting, but is it a request ?
```

```
        If oMtg.MeetingType <> g_Const_CDOMeetingRequest then
```

```
            Call DebugAppend("Meeting item is not a request",False)
```

```
        Else
```

```
            Set oAppt = oMtg.GetAssociatedAppointment
```

```
            Resp = oUser.GetFreeBusy(oAppt.starttime, oAppt.endtime, 30)
```

```
            Call DebugAppend("Free/busy times = " & Resp,False)
```

```

    If Resp = g_Const_CdoFree then
        Call DebugAppend("Free - Accepting meeting",False)
        Set oMtgResp = oMtg.Respond(g_Const_CDOResponseAccepted)
        oMtgResp.Text = "Conf room available. Meeting accepted."
    Else
        Call DebugAppend("Declining meeting",False)
        Set oMtgResp = oMtg.Respond(g_Const_CDOResponseDeclined)
        oMtgResp.Text = "Conf room unavailable for selected time. Meeting
declined."
    End If

    oMtgResp.Send

    End If
End If

Call DebugAppend("Error Detected: ",True) 'Check for any possible sys errors

Script.Response = g_bstrDebug

```

```
End Sub
```

```
' DESCRIPTION: This event is fired when a message in the folder is changed
Public Sub Message_OnChange
```

```
'Not Used
```

```
End Sub
```

```
' DESCRIPTION: This event is fired when a message is deleted from the folder
Public Sub Folder_OnMessageDeleted
```

```
'Not Used
```

```
End Sub
```

```
' DESCRIPTION: This event is fired when the timer on the folder expires
Public Sub Folder_OnTimer
```

```
'Not Used
```

```
End Sub
```

```
'-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
'
'                PRIVATE FUNCTIONS/SUBS
'
'+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

```
'-----
' Name: DebugAppend
' Area: Debug
' Desc: Simple Debugging Function
' Parm: String Text, Bool ErrorFlag
'-----
```

```
Private Sub DebugAppend(bstrParm,boolErrChkFlag)
```

```
    if boolErrChkFlag = True then
        if err.number <> 0 then
            g_bstrDebug = g_bstrDebug & bstrParm & "-" & cstr(err.number) &
err.description & vbCrLf
        end if
    end if

```



```
        err.clear
    end if
else
    g_bstrDebug = g_bstrDebug & bstrParm & vbCrLf
end if
```

End Sub

</SCRIPT>

Using AutoAccept

Group

To use the AutoAccept sample application

1. Log on to Microsoft Outlook as yourself.
2. Using Microsoft Outlook, create a meeting request in which you specify the conference room whose mailbox is being monitored by AutoAccept. You can start by clicking **Plan a Meeting** on the **Compose** menu. For more information, see the topic "Meeting Planner" in the online Help for Microsoft Outlook.
3. Send this meeting request to the mailbox of the conference room.

If the room is free for the time you requested, the AutoAccept script grants your request and enters meeting information into the calendar of the conference room. It also sends to all meeting participants a message indicating that the meeting request has been accepted, and invites them to the meeting.

Notes The AutoAccept sample application has minimal error checking. If you want, you can create a more robust application by adding more error checking to the script, `autoaccept.txt`.

The AutoAccept sample application does not handle recurring appointments, nor can it remove meetings if a cancellation request is received. AutoAccept can only accept or deny meeting requests outright; it cannot accept a meeting request tentatively.

BankPost Sample Application

BankPost is a sample application implemented in an event script that is triggered by the **Folder_OnMessageCreated** event. You install this script in the Inbox folder of a "Bank Teller" recipient. By calling Microsoft® Transaction Server (MTS) objects, this script posts money to a sample bank account when new mail arrives in the folder.

Installing the BankPost Agent

Group

To install the BankPost agent

1. Install Microsoft Transaction Server (MTS) on a Microsoft® Exchange Server computer.
2. In the online Help for MTS, locate the example called **Validating Setup with a SQL Server Transactional Component**. Follow the steps of this example, which helps you install Microsoft® SQL Server version 6.5 and the "Sample Bank," both of which are needed to run the BankPost agent. Be sure to complete all the setup procedures in this example.
3. Create a mailbox (suggested name: "Bank Teller") on the Microsoft Exchange Server computer.
4. Using the Microsoft Exchange Server Administrator program, grant the Bank Teller's mailbox the permission needed to install scripts—for example, Author permission. For more information, see "Granting Permissions" in the *Microsoft Exchange Server Programmer's Reference* in the Microsoft® Platform SDK documentation.
5. Give yourself a "Manager" role for the Sample Bank. To do this, open the Microsoft Transaction Server Explorer; then, in the hierarchy under **My Computer/Packages Installed/Sample Bank/Roles/Managers/Users**, right-click and add your own Microsoft® Windows NT® account name. If you are a "Manager," you can add \$500 or more in a single transaction using this sample application. If you are not a "Manager," this sample application displays an error message for transactions of \$500 or more.

Alternatively, find this location in the hierarchy, click **File** on the **Users** menu, click **New**, and then add your Windows NT account name.
6. Start Microsoft Outlook and locate the mailbox for the Bank Teller whose events you want the BankPost script to monitor.
7. Right-click this mailbox, or click **Folder** on the **File** menu.
8. Click **Properties** for that folder.
9. Click the **Agents** tab to display the **Agents** property page. (This tab is not displayed unless the Microsoft Exchange Event Service is installed properly).
10. Click **New**. This displays the **New Agent** dialog box.
11. To enable monitoring of new messages, select the check box **A new item is posted in this folder**.
12. Click **Edit Script**.
13. Paste the BankPost script code, which you can find in [BankPost Files](#).
14. Close all dialog boxes by clicking **OK** in each. The BankPost agent is now installed.

BankPost Files

This sample application includes only a single file, the BankPost script. This script, called BankPost.txt, is provided here. You can use it in the installation of this sample application, described in [Installing the BankPost Agent](#):

```
<SCRIPT RunAt=Server Language=VBScript>
```

```
'THIS CODE AND INFORMATION IS PROVIDED "AS IS" WITHOUT  
'WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED,  
'INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTIES  
'OF MERCHANTABILITY AND/OR FITNESS FOR A PARTICULAR  
'PURPOSE
```

```
'-----  
' FILE DESCRIPTION: Bank update sample for Exchange Scripting Agent  
'  
' FileName: BankPost.Txt  
'  
' Copyright (c) Microsoft Corporation 1993-1998. All rights reserved.  
'-----
```

Option Explicit

```
'-----  
' Global Variables  
'-----
```

```
Dim g_bstrDebug
```

```
'-----  
' Event Handlers  
'-----
```

```
' DESCRIPTION: This event is fired when a new message is added to the folder  
Public Sub Folder_OnMessageCreated
```

```
on error resume next
```

```
Dim oBank      'Bank Object  
Dim oMsg       'Message Object  
Dim Item       'Item Object
```

```
Dim bstrPrimeAcct 'Account Number String  
Dim bstrAmount    'Account Number Amount  
Dim bstrTempVar   'Temp Variable String
```

```
Dim varRes      'Result Variant
```

```
'Get the incoming message object  
Set oMsg = EventDetails.Session.GetMessage(EventDetails.MessageID, Null )  
Call DebugAppend("Get Message", True)
```

```

'Get Bank Object and Msg Information
set oBank = CreateObject("Bank.Account.VC")
  bstrPrimeAcct= oMsg.Subject
  bstrAmount = oMsg.Text

'Post to the account
varRes = oBank.Post(CLng(bstrPrimeAcct),CLng(bstrAmount),CStr(bstrTempVar))
Call DebugAppend("oBank.Post",True)

'Display Results
if varRes = 0 then
  Call DebugAppend("OK: " & bstrAmount & " posted to bank account " &
bstrPrimeAcct,False)

else
  Call DebugAppend("ERR: Manager role required for amounts over $500.",False)
end if

Call DebugAppend("Script",True) 'Trap any error and show info
script.response = g_bstrDebug

```

End Sub

```

' DESCRIPTION: This event is fired when a message in the folder is changed
Public Sub Message_OnChange

```

```

  'Not Used

```

End Sub

```

' DESCRIPTION: This event is fired when a message is deleted from the folder
Public Sub Folder_OnMessageDeleted

```

```

  'Not Used

```

End Sub

```

' DESCRIPTION: This event is fired when the timer on the folder expires
Public Sub Folder_OnTimer

```

```

  'Not Used

```

End Sub

```

'-----
'                                     PRIVATE FUNCTIONS/SUBS
'-----

```

```

'-----
' Name: DebugAppend
' Area: Debug
' Desc: Simple Debugging Function
' Parm: String Text, Bool ErrorFlag
'-----

```

```

Private Sub DebugAppend(bstrParm,boolErrChkFlag)

```

```

  if boolErrChkFlag = True then
    if err.number <> 0 then
      g_bstrDebug = g_bstrDebug & bstrParm & " Failed: " & cstr(err.number) &
err.description & vbCrLf

```

```
        err.clear
    end if
else
    g_bstrDebug = g_bstrDebug & bstrParm & vbCrLf
end if
```

End Sub

</SCRIPT>

Using BankPost

This sample application uses the subject field of the sent or posted message as the bank account number. The number "1" is used as the default bank account number although you can change it to any arbitrary number. The message body contains the amount of money, specified using an integer value. In other words, use whole dollars and no cents.

Testing the BankPost Agent

Group

To test the the BankPost agent

1. Log on to Microsoft Outlook as Bank Teller.
2. To your own Inbox, post a message whose subject field contains the number "1" and whose message body contains the number "20".
3. View the Windows NT Event Log or use the Microsoft Transaction Server administrator program to verify success or failure.
4. To see the total number of dollars in the Sample Bank, run it using its Microsoft® Visual Basic® interface.

Notes The BankPost script calls the Microsoft® Visual C++® version of the Bank.Account object, but it could just as easily use the Visual Basic or Java version of the object. Also, this sample script has very little error checking. A more robust application would perform extensive error checking on numbers such as the account and amount data submitted.

This script uses the Sample Bank from Microsoft SQL Server version 6.5; other versions of Microsoft SQL Server may have different arguments for the **Post** method of the **IDispatch** object.

Microsoft Exchange Discussion Wizard for FrontPage

With the Microsoft® Exchange Discussion Wizard for Microsoft® FrontPage®, you can quickly create a discussion Web site, or add discussion to an existing Web site, without writing any Active Server Pages (ASP) or HTML script.

The Microsoft Exchange Discussion Wizard differs from the basic discussion wizard included with Microsoft FrontPage in that it stores messages and documents on a Microsoft Exchange Server computer. This provides the following benefits:

- Because discussion data is stored in a Microsoft Exchange information store, both Web users and Microsoft® Outlook™ users can participate in the same discussion and share information.
- You can create views on Microsoft Exchange folders that are also available to both Microsoft Outlook and Web users.

Discussion Wizard Software Requirements

The client computer on which you use the Microsoft Exchange Discussion Wizard requires the following software:

- Microsoft® Windows NT® version 4.0 or later, or Microsoft® Windows® 95
- Microsoft FrontPage 97 or FrontPage 98
- Microsoft Outlook version 8.01 or later, or the Microsoft Exchange client version 5.0 or later

The Web server on which you host the discussion requires the following software:

- Microsoft Windows NT version 4.0 or later
- Microsoft® Internet Information Server (IIS) version 3.0 with Active Server Pages and with FrontPage server extensions, or IIS version 4.0 (when available)
- Microsoft Outlook Web Access. You can install this by using the setup program for Microsoft Exchange Server.

Installing the Microsoft Exchange Discussion Wizard

Group

To install the Microsoft Exchange Discussion Wizard

On the client computer on which you will use the Microsoft Exchange Discussion Wizard:

1. Run the discwiz.exe program in the \server\support\collab\fpwiz subdirectory on the Microsoft Exchange Server 5.5 CD-ROM. This expands the wizard's files into a temporary directory on your computer.

- Or -

Connect to the Microsoft Exchange Application Farm site at <http://www.microsoft.com/technet/appfarm>, locate the Microsoft Exchange Discussion Wizard for FrontPage, and follow the download instructions.

2. If a dialog box appears that advises you to shut down other applications, please do so. This avoids problems related to updating shared files. When you have done this, click **OK**.
3. If you were previously using the beta version of this wizard, setup asks whether you want to remove the older version. Click **Yes**.
4. In the **Front Page Exchange Discussion Wizard Setup** dialog box, click the large button to continue setup. Setup then installs the wizard.

Using the Microsoft Exchange Discussion Wizard

Group

To create a new Web that contains a Microsoft Exchange discussion

1. Using Microsoft Outlook or the Microsoft Exchange client, locate the public folder you want to use for this discussion.
If the folder does not yet exist, create it before running the wizard.
If you want anonymous Web users to have access, be sure to grant "Anonymous" access to the public folder. For more information on setting public folder permissions, consult the Help file for Microsoft Outlook or for the Microsoft Exchange client.
2. Start Microsoft FrontPage.
3. If the **Getting Started** dialog box is displayed when you start FrontPage, select **Create a New FrontPage Web** and click **OK**.
If the **Getting Started** dialog box is not displayed, click **New FrontPage Web** on the **New** submenu on the **File** menu.
4. In the **New FrontPage Web** dialog box, answer the following questions:
 - In question 1, choose **From Wizard or Template** and select **Microsoft Exchange Discussion Wizard**.
 - In question 2, name your new Web.
Click **OK**.
5. A **Choose Profile** dialog box may be displayed. If so, select the profile you wish to use, and click **OK**.
6. Select the public folder you would like to use for the discussion. After you have selected a folder, click **Next**.
7. The wizard adds a link to your discussion on an existing page in your site, or creates a new page if you prefer. Enter the following three items:
 - **Name of the discussion.** This is the hyperlink that users click to move to the discussion. **Note** If you are using Microsoft FrontPage 97, make a note of this discussion name; you will need to use it in step 11.
 - **Description of the discussion.** This appears after the underlined text.
 - **Destination page for the link.** If this page already exists, the wizard adds the link to the page. If the page does not already exist, the wizard creates it for you and adds it to your Web.When you have finished, click **Next**.
8. Choose whether to require authenticated access or to allow anonymous access to the discussion site.
Also configure IIS and Microsoft Exchange Server for authenticated or anonymous access to match your choice on this screen. For more information, see [Configuring for Anonymous Access](#) or [Configuring for Authenticated Access](#).
When you have finished, click **Next**.
9. Click **Finish** to create the Web.
10. When the Microsoft FrontPage Editor window displays the page into which the discussion link was inserted, confirm that the wording and formatting is to your liking. If you would like to make changes, use the Editor to make changes. Then click **Save** on the **File** menu to post the change to your Web.
If you are using Microsoft FrontPage 97, you need to manually configure the discussion folder to allow scripts to be run. To do this:
 - In the Microsoft FrontPage Explorer, select the subfolder containing the discussion scripts. This

subfolder has the discussion name you selected in step 7.

- Click **Properties** on the **Edit** menu.
- In the **Properties** dialog box, select **Allow scripts to be run**. Click **OK**.

You can use this new discussion Web by connecting your browser to `http://<servername>/<webname>/<destpage>`. <Destpage> is the destination page that you specified in step 7.

Group

To add a Microsoft Exchange discussion to an existing Web

Follow the procedure **To create a new Web that contains a Microsoft Exchange discussion** (see [Using the Microsoft Exchange Discussion Wizard](#)), but with the following modification:

4. In the **New FrontPage Web** dialog box, answer the following questions:
 - In question 1, choose **From Wizard or Template** and select **Microsoft Exchange Discussion Wizard**.
 - Select the **Add to Existing Web** check box.

Click **OK**.

Configuring for Authenticated Access

Authenticated access means that users are required to present a user name and password to participate in the discussion. There are three configuration steps you should take to set up an authenticated discussion Web:

Group

To configure for authenticated access

1. Choose authenticated access in the wizard when you create the discussion Web.
2. In IIS version 3.0, use the Microsoft Internet Service Manager to configure which authentication types are accepted by IIS. For more information, see [IIS Authentication](#).
3. In Microsoft Outlook or the Microsoft Exchange client, configure which authenticated users have access to the public folder that hosts the discussion. For more information, see [Microsoft Exchange Server Authentication](#).

IIS Authentication

This wizard supports two kinds of authentication:

- **Basic authentication.** This technique is available for any discussion site. When users connect to the discussion Web, their browser sends user name and password to the IIS computer. For maximum security, it is recommended that you combine this technique with Secure Sockets Layer (SSL); for more information about SSL, consult the *IIS Installation and Administration Guide*.
- **NTLM authentication.** This technique is available only if the Microsoft Exchange Server hosting your discussion is running on the same computer as IIS. When users connect to the discussion Web, their browser uses an encrypted protocol to send authentication information to the IIS computer.

Microsoft Exchange Server Authentication

After users have been authenticated, Microsoft Exchange Server can determine whether they have permission to use the public folder that hosts this discussion. To participate in the discussion, a user must have Read Items and Create Items permission on the folder. Use Microsoft Outlook or the Microsoft Exchange client to set folder permissions for individual users or groups of users.

For related information, see Chapter 6, "Configuring Public Folders," in the *Microsoft Exchange Server Getting Started Guide*.

Group

To configure IIS authentication type

1. Click the Windows **Start** button and click **Programs**. Now click **Microsoft Internet Server** and click **Internet Service Manager**.
2. In the main window of the Internet Service Manager, choose the computer running IIS version 3.0. Then click **Service Properties** on the **Properties** menu.
3. On the **Service** page
 - To enable Basic authentication, select the **Basic (Clear-Text)** check box.
 - To enable NTLM authentication, select the **Windows NT Challenge/Response** check box.
4. Click **OK**.

Configuring for Anonymous Access

Anonymous access means that any user will be permitted to participate in the discussion.

You decide whether a discussion permits anonymous access when you create the discussion Web using the wizard.

When users post to the discussion they will be required to enter their e-mail address in the form "user@server.domain" for identification. **Note** This e-mail address is not verified in any way, so this feature should be considered a convenience but is not secure. If security is a concern, we recommend that you require authenticated access.

Group

To configure IIS version 3.0 to permit anonymous access

1. Click the Windows **Start** button and click **Programs**. Now click **Microsoft Internet Server** and click **Internet Service Manager**.
2. In the main window of the Internet Service Manager, choose the computer running IIS version 3.0. Then click **Service Properties** on the **Properties** menu.
3. On the **Service** page, select the **Allow Anonymous** check box. Click **OK**.

Group

To grant anonymous access to a public folder

1. Log on to Microsoft Outlook or the Microsoft Exchange client.
2. Select the public folder that contains messages and documents for the discussion.
3. On the **File** menu, click **Folder**, and then click **Properties**.
4. In the folder **Properties** dialog box, click the **Permissions** tab.
5. In the **Name** list box, select the line for Anonymous.
6. In the **Roles** drop-down list box, click **Author**.
7. Click **OK**.

For related information, see Chapter 6, "Configuring Public Folders," in the *Microsoft Exchange Server Getting Started Guide*.

Microsoft Exchange Form Design Wizard Sample

With the Microsoft® Exchange Form Design Wizard you can quickly create customized HTML forms based on the built-in forms included with Microsoft® Outlook™ Web Access. Additional layout and formatting enhancements can then be done in an HTML/ASP editor such as Microsoft® FrontPage®. Additional scripting enhancements can be done with a script editor such as Microsoft® Visual InterDev™.

The following types of form are supported by the wizard:

- Forms used to post information directly to public folders (based on the item type IPM.Post)
- Forms used to send information to other recipients (based on the item type IPM.Note)

Form Design Wizard Software Requirements

The computer on which you run the Form Design Wizard requires the following software:

- Microsoft® Windows NT® version 4.0 or later

The Web server on which you post the resulting form requires the following software:

- Microsoft Windows NT version 4.0 or later
- Microsoft® Internet Information Server version 3.0 or later (be sure to install Active Server Pages)
- Microsoft Outlook Web Access included with Microsoft Exchange Server version 5.5 or later

The server hosting Microsoft Exchange Server requires the following software:

- Microsoft Exchange Server version 5.0 (together with Service Pack 1) or later. **Note** If this is the same computer as the Web server, you must run Microsoft Exchange Server 5.5, to match Outlook Web Access 5.5, listed previously)

Installing the Form Design Wizard Sample

Group

To install the Microsoft Exchange Form Design Wizard

1. Run the fdsetup.exe program in the \server\support\collab\sampler\formwiz subdirectory on the Microsoft Exchange Server 5.5 CD-ROM. This expands the wizard's files into a temporary directory on your computer, and starts the main setup program. **Note** If updates to this wizard become available, they will be posted to the Microsoft Exchange Application Farm Web site at <http://www.microsoft.com/technet/appfarm>.
2. If a dialog box appears that advises you to shut down other applications, please do so. This avoids problems related to updating shared files. When you have done this, click **OK**.
3. In the **Form Design Wizard Setup** dialog box, confirm the target directory. To change the target directory, click **Change Directory**. In the **Change Directory** dialog box, select the directory you want and click **OK**.
4. In the **Form Design Wizard Setup** dialog box, click the large button to install the wizard.

Using the Form Design Wizard Sample

Before you start designing a form, take these steps:

Group

To prepare for using the Form Design Wizard Sample

1. Locate or install Microsoft Outlook Web Access included with Microsoft Exchange Server version 5.5 on a Web server to which you have write permissions. If you need to install Microsoft Outlook Web Access, you can do so from the main Microsoft Exchange Server setup utility.
2. If you plan to run the wizard on a computer other than the Web server containing the Microsoft Outlook Web Access scripts, you must share the scripts directory so that the wizard can copy your custom form to the server. Use the Microsoft Windows NT Explorer or a similar utility to ensure that the **webdata** directory (for example: c:\exchsrvr\webdata) is shared. Record this share name so you can supply it when you run the wizard.

Group

To run the Form Design Wizard sample

Click **Form Design Wizard** on the **Programs** submenu on the Windows **Start** menu.

The Resulting Form

When you have finished, the wizard will have created a custom form for you in the Outlook Web Access server you indicated on the initial pages of the wizard.

Under the **webdata** tree of Outlook Web Access, the form is copied to a location that depends on the item type you selected in the wizard. For example, if you choose to create a custom status form of type IPM.Note.Status, the form scripts are copied to **webdata/usa/forms/ipm/note/status**.

The form consists of the following files:

| File Name | | Description |
|-----------------------------|---------------------------|---|
| Note-based forms | Post-based forms | |
| frmRoot.asp | frmRoot.asp | This is the main entry point of the form. Every custom HTML form has at minimum a frmRoot.asp form, although each form's frmRoot.asp script may be different.
For forms created by this wizard, frmRoot.asp performs the following: <ul style="list-style-type: none">• Determines whether an existing item is being opened (read mode) or a new item created (compose mode).• Creates the appropriate frameset, depending on which mode applies. The "cmp-" prefix indicates files used in compose mode. If files prefixed with "red-" exist, they are used in read mode; otherwise, files with the "cmp-" prefix are used for both modes.• In read mode, frmRoot.asp retrieves the existing message using Collaboration Data Objects (CDO). This message object is then cached for use by the other scripts in this form.• Provides utility functions for use by the remaining scripts. |
| form.ini | form.ini | Plain-text INI file that provides a friendly display name with which users can choose this form after it is deployed. This file can optionally mark a particular form as hidden, so it cannot be composed but can be used to read existing items. An example is a meeting response form that is not created by the Compose New command, but is created by clicking Accept / Decline / Tentative on a meeting request form. |
| cmpTitle.asp / redTitle.asp | PostTitl.asp | This is the topmost page you can see on the form. It implements the toolbar and tab strip. |
| cmpMsg.asp / redMsg.asp | postMsg.asp / redPost.asp | This file provides content of the first page. By default this page is titled "Message" but this can be customized in the wizard. |
| cmpOpt.asp / redOpt.asp | (not available) | This file provides the content of the Options page. This page can be disabled in the wizard. |
| cmpAtt.asp | postAtt.asp | This file provides a way to attach files while composing a message. This page can be disabled in the wizard. |
| page_N.asp / pageR_N.asp | page_N.asp / pageR_N.asp | There can be up to five of these files, each of which provides the content of your custom form pages. The custom content on the first page is handled by |

| | | |
|--------------|--------------|--|
| commands.asp | commands.asp | cmpMsg.asp/redMsg.asp mentioned previously.
This is a utility script used for server-side command handling; for example, checking names in the To: field against the Microsoft Exchange directory when you click the Send button on the toolbar. |
| delete.asp | delete.asp | This is another utility script used to delete an item if you click the Delete button on the toolbar. |
| *.gif | *.gif | Miscellaneous images. "Invisibl.gif" is used for spacing; "Next.gif" and "Last.gif" provide button faces for the Next and Previous toolbar buttons. |
| wizard.inc | wizard.inc | Include file containing various constants used by the various form scripts. |

What this Wizard Does Not Do

Because it is a sample, this wizard does not do everything you might expect from a form-customization tool. Please consider the following items:

- **Test, test, and test again.** There is wide variation between the HTML support available in the currently popular browsers. You are strongly encouraged to test your custom forms on the browsers your users will use.
- **Data validation.** This wizard does not perform explicit data validation on custom fields, nor inform users of errors. It will ensure that only correct data types are bound to the message, but will fail silently for invalid data, for example if you type "test" into a date field.
- **Full range of predefined fields.** The wizard focuses on allowing you to place custom fields on your forms, and does not include the full range of predefined fields present in the form designer of Microsoft Outlook.

Microsoft Outlook Web Access

With Microsoft® Outlook™ Web Access, users can access data on a Microsoft® Exchange Server computer using an Internet browser (that supports frames and JavaScript) from a UNIX, Macintosh, or Microsoft® Windows®-based computer. Web Access provides Web-based public access to Microsoft Exchange Server public folders and the global address list. Authenticated users can log on to their personal accounts to read private e-mail and send messages. Using Web-based public folder access, an organization can build private and public discussion forums on the Internet and private intranets. Users can publish information on the Internet without having to manually convert documents to HTML format.

With an Internet browser, users specify the Uniform Resource Locator (URL) using HTTP to access the virtual root for the Microsoft Exchange Server computer. Then, after providing the proper logon credentials, they can access their mailbox and public folder data. Microsoft Exchange Server data is translated by Active Server Pages into HTML and transmitted to the browser using HTTP.

Browser access to Active Server Pages through HTTP is controlled through the Microsoft Exchange Administrator program. Access can be granted on a per-user basis or made available to any user through anonymous access.

Microsoft Outlook Web Access uses files of these types:

- **.asp files** Contain Active Server Pages scripts.
- **.htm files** HTML files that do not contain Active Server Pages scripts.
- **.gif files** Graphic image files used to depict items such as screen titles, buttons, and icons.

Installing Microsoft Outlook Web Access

Group

To install Microsoft Outlook Web Access

1. Run server setup on the distribution compact disk of Microsoft Exchange Server.
2. Select **Complete/Custom**.
3. Select the **Outlook Web Access** option. This installs the CDO Library, the CDO Rendering Library, and Outlook Web Access .asp files.

Common Tasks

These are some of the most common actions performed with Microsoft® Outlook™ Web Access:

- An authenticated user opens a mailbox.
- A user (authenticated or not) accesses a public folder.
- An authenticated user reads a message in a mailbox.
- An unauthenticated user reads a message in a public folder.
- An authenticated user sends a message.
- A user (authenticated or not) posts a message to a public folder.
- A user (authenticated or not) looks for recipients in the address book.

Some of these tasks are explained in the [following topic](#).

Note From the server, your CDO application cannot view the contents of a personal folder store (PST). Likewise, you cannot view the contents of a personal address book (PAB). Personal folder stores and personal address books are accessible when the CDO application is run on the client workstation.

Sequence of Events

The following procedures describe the events that take place and the flow of data as selected messaging tasks are performed. To see a diagram showing how the elements mentioned in these descriptions relate to one another, see [The Active Server Components of Microsoft Exchange](#).

Group

An authenticated user opens a mailbox

1. A user clicks the logon URL or follows a link (<http://<server>/exchange>, a virtual root) to the logon page for Active Server™ components.
2. The user enters a mailbox name, domain account name, and password. The script for the logon interaction is contained in the `logon.asp` file.
3. If the user is authenticated through the Microsoft® Windows NT® domain controller, an Active Server Pages session is started, which maintains session information until the user logs off or the session times out. (After authentication, the session timer starts when the first page is sent to the browser, and is restarted when each subsequent page is sent to the browser.)

A CDO Session object is created and stored as a state variable in the Active Server Pages session. This lets the CDO Library and CDO Rendering Library make messaging calls to retrieve information store data from Microsoft® Exchange Server.

4. The Inbox for this Session object is opened. This becomes visible to the user as the script file `root.asp` is executed. The Inbox is displayed in four frames from the `\inbox` directory. (Anonymous users are directed to the file `\anon\root.asp`.) For more information, see [Directory Structure](#).

Group

An unauthenticated user accesses a public folder

1. A user clicks the logon URL or follows a link (<http://<server>/exchange>, a virtual root) to the logon page. (Alternatively, a user may have followed a URL to the public folder, in which case the user is logged on transparently, and sees no logon page.)
2. The user clicks **Public Access** (or the appropriate **Click here**) on the logon screen.
3. A CDO session is started for the unauthenticated user.
4. The root folder of the public folder tree is displayed as the file `\anon\root.asp` is rendered. Only certain public folders are visible. The administrator determines which folders are available to unauthenticated users, using the HTTP Protocol object in the Protocols container on the site.

Group

An unauthenticated user posts a message to a public folder

1. The user opens the public folder, as described in the previous procedure.
2. The user clicks **Post New Item**. The browser supplies the name "Anonymous" (or a suitable equivalent in the user's language, if not English) to be displayed on messages posted to public folders. Unauthenticated users can post messages, but they cannot send e-mail messages.

Group

A user reads a document or message in a public folder

- Or -

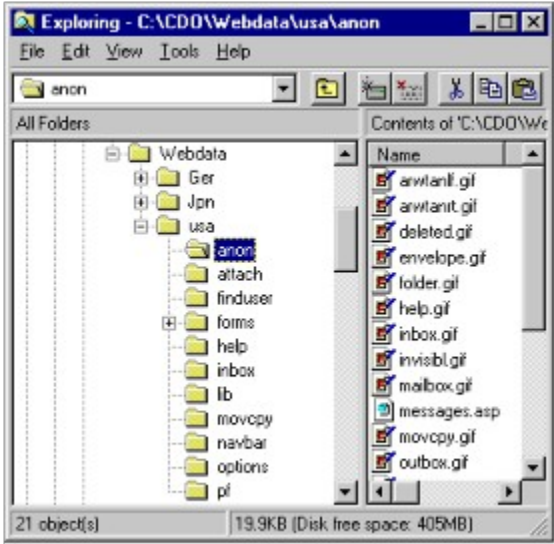
Group

An authenticated user reads a message in a mailbox or public folder

1. The user opens the mailbox or public folder, as in one of the previous procedures.
2. The user clicks a message in the mailbox or public folder.
3. If the user is opening a message in a mailbox, the file `\forms\ipm\note\read.asp` is loaded into Microsoft® Active Server Pages, which renders the message using the CDO Rendering Library. (The file `\forms\ipm\post\read.asp` is used to render a message in a public folder.)

Directory Structure

After the Microsoft® Exchange Active Server™ components are installed on a server, the files of Microsoft® Outlook™ Web Access reside in a directory structure similar to that in the following illustration. A directory tree exists for each language you install; in the illustration, the usa tree is expanded, and some of the files in the usa\anon subdirectory are listed in the right pane.



The ASP files used by Microsoft Exchange Active Server components reside in these subdirectories. For example, the \inbox directory contains .asp files used by the Active Server components to display a user's mailbox and its contents. Other subdirectories contain files for other uses; for example the \anon directory contains files that are rendered to show public folders and their contents to an unauthenticated user, whereas the files in \pf are used when an authenticated user accesses public folders.

The \forms directory and its subdirectories contain scripts for every form. The structure of the \forms directory is based on the message class of the form; the .asp files in each directory are named after the verb that is applied – such as read.asp and compose.asp. For example, an IPM.Note is read using the file read.asp in the directory \forms\ipm\note. Customizers can install additional forms for each message class they define using the same structure. This structure is loaded at application start.

Handling Multiple Languages

Microsoft® Outlook™ Web Access handles the rendering of different languages in the following way:

- Scripts for all languages are installed under the same virtual root (typically //<server>/exchange), in a separate directory for each language. See the illustration in [Directory Structure](#), which shows directories for German (Ger) and Japanese (Jpn) as well as USA-English (usa). Additional language packs can be obtained to support other languages.
- An Internet Server API (ISAPI) filter, ExchFilt.dll, is installed. This filter examines all incoming URLs. If a URL is for one of the applications defined in \\HKLU\System\CurrentControlSet\Services\MSExchangeWeb\Applications (Microsoft® Exchange Server is defined by default, but any application can be added to this list), the filter will:
 1. Examine the AcceptLanguage header from the browser. (The AcceptLanguage is the language stated to be preferred by a given browser.)
 2. Look up the header using the table laid out in \\HKLU\System\CurrentControlSet\Services\MSExchangeWeb\AcceptLanguages.
 3. Insert the directory name into the URL after the application name.

For example, if you have a browser that prefers USA-English, //MyServer/Exchange/logon.asp becomes //MyServer/Exchange/USA/logon.asp.

This process enables a Web browser being used with Microsoft Exchange Active Server™ components to run the set of scripts most suitable for a particular user's language. In the case where a particular language is not installed, the default will be the language of the Microsoft Exchange Server computer.

Note that code page and locale are also defined in these registry settings and can be used to refine support for specific sublanguages.

Impersonation

A user's access to Microsoft® Exchange Server information is handled in a thread of execution within the Microsoft® Internet Information Server (IIS) process. If the user wants authenticated access – to open a mailbox, for example – this thread must impersonate a Microsoft® Windows NT® security context. In other words, to be granted authenticated access to the Microsoft Exchange information store, a thread must be associated with a set of valid security credentials.

The impersonation process has two parts:

1. At the time a user logs on, save the valid security context into the Session object.
2. When rendering a page in a multiframe set, or when a session ends, retrieve the saved security context and call the **Impersonate** method on the RenderingApplication object.

Group

To save a security context

At the time of user logon, use a command such as the following (from the file lib\logon.inc), to save the current security context in the session object:

```
Session("hImp") = objRenderApp.ImpID
```

Group

To impersonate the logged on user

1. When rendering a page in a multiframe set, or as a session is ending (such as in the method **Session_onEnd** in the file global.asa), retrieve the saved security context handle from the Session object. The following code is from the file lib\session.inc:

```
hImp = Session("hImp")
```

2. Get the RenderingApplication object and call the **Impersonate** method, passing the security context handle, as shown in the following code from the file lib\session.inc.

```
set objRA = Application(bstrRenderApp)  
objRA.Impersonate(hImp)
```


The Frameset Design-Time Control

The Frameset Design-Time Control (DTC), for use within Microsoft® Visual InterDev™, lets you quickly develop Web-based applications based on data from Microsoft® Exchange, including information-store, calendar, and address-book data. The Frameset DTC uses Microsoft® Outlook™ Web Access scripts, but it also lets you design Web pages that are customized to meet your needs. This section describes the features in the Frameset DTC and explains how to use them.

Microsoft Exchange Server version 5.5 includes a rich Web-based e-mail client known as Microsoft Outlook Web Access. Using Outlook Web Access, users can browse messages in their personal folders or in a public folder. Outlook Web Access also gives users access to their Outlook calendar.

Outlook Web Access is based on Microsoft® Internet Information Server (IIS), Active Server Pages (ASP), and Microsoft® Collaboration Data Objects (CDO). Using the Frameset DTC, you can build your own Web-based applications using the existing Outlook Web Access ASP files.

Installing Outlook Web Access and the Frameset DTC

Web pages that are built using the Frameset DTC require Microsoft® Outlook™ Web Access to be installed on a Microsoft® Internet Information Server. You can install Outlook Web Access by running Microsoft® Exchange Server setup and selecting the Outlook Web Access option. Outlook Web Access does not need to be installed on the same computer as Microsoft Exchange Server.

You use the file FSSETUP.EXE to install the Frameset DTC and register its OCX file. This installation allows you to run the DTC within Microsoft® Visual InterDev™.

Note The server onto which you install Outlook Web Access becomes your Outlook Web Access server, and the name of this server is required when using the Frameset DTC.

Using the Frameset DTC

The Frameset DTC is designed to be used in design environments that support DTCs, including Microsoft® Visual InterDev™. Using Visual InterDev, you can insert Design Time Controls on any HTML or ASP page.

To insert a DTC using Visual InterDev, select the **Into HTML** option from the **Insert** menu, and then select **ActiveX Control**. Next select the **Design-time** tab. A list of DTC controls is displayed. When you select **FrameDTC.Frameset** from the list, this DTC is inserted into the current HTML page, and the control's property page is displayed. When the DTC is closed, it automatically generates the necessary code and inserts the code into the Web page you are editing.

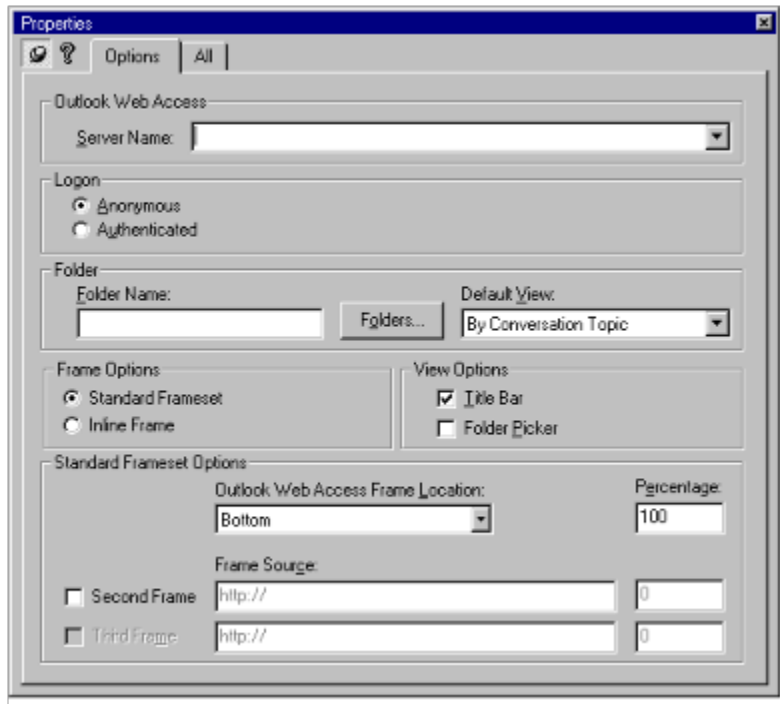
The DTC control is displayed as green text in Microsoft Visual InterDev. To change its properties, select the green text, right-click the selection, and press **Edit Design-Time Control**. Again, the properties for the DTC are displayed. The code the DTC generates is updated when you close the DTC.

You must remove the HTML <BODY> and </BODY> tags or insert the Frameset DTC in a location outside these BODY tags for the page to display the frames and content correctly. This restriction does not apply when using inline frames, which are a feature present in Microsoft® Internet Explorer version 3.0 and above.

Frameset DTC Properties

You use the **Properties** property sheet of the Frameset DTC to control its functionality. On the **Options** page of the Frameset DTC property sheet, properties are grouped under the group boxes **Outlook Web Access**, **Logon**, **Folder**, **Frame Options**, **View Options**, and **Standard Frameset Options**.

These options are visible in the following illustration:



Outlook Web Access Group Box

The only setting to make in the **Outlook Web Access** group box is in the text box **Server Name**. Here you provide the Frameset DTC with the name of the Outlook Web Access server. This should be the name of the server only, and need not include the word \Exchange.

For example, do not type:

```
\\<servername>\Exchange
```

Instead, simply type:

```
<servername>
```

Logon Group Box

This option lets you set the type of access (anonymous or authenticated) that people have after they log on to Outlook Web Access. To make this choice, click **Anonymous** or **Authenticated**.

With anonymous access, users are not prompted for logon information or their Microsoft® Windows NT® credentials, and they have access only to the public folders that have had "anonymous" permissions enabled. Anonymous logon provides the ability to make a folder accessible to people who are not defined within the Microsoft® Exchange organization. A typical example of this is a folder that acts as a repository for a newsgroup or a discussion and has been made accessible on the Internet, where prospective users are not known by the person hosting the discussion.

For the authenticated option, users are prompted to provide credentials when they log on, and they can access any folder they have permissions to. Use this option if you want users to be able to access their Inboxes. You can also use this option to make a public folder accessible only to an explicit list of users who have already been defined within the Microsoft Exchange organization. Additionally, these users can be given a highly granular level of permission to the particular folder. For example, some might be given Author permission while others are given only Reader permission.

Folder Group Box

The options in the **Folder** group box determine which folder's contents are displayed, and how they are displayed.

The **Folder Name** field determines what folder contents are displayed when the Web page is viewed. To select a folder, press the **Folders** button. If this is the first time you select a folder, you are prompted for the name of your Microsoft Exchange profile. After you have selected the profile, a list of folders is displayed. Select the folder you want and click **OK**. Your Microsoft Exchange profile information is saved; if you later need to use another profile, click the **Choose Profile** button.

When folder contents are displayed, you can set the default view by selecting the view from the **Default View** list box. Selecting a view determines how messages are viewed.

Frame Options Group Box

The **Frame Options** control lets you choose between displaying folder contents in a standard frame or in an inline frame. Selecting **Inline Frame** displays the messages inline with other content on the Web page. If you select **Standard Frameset**, the **Standard Frameset Options** are enabled, which lets you choose to have one, two, or three frames displayed.

View Options Group Box

The **View Options** group box contains the following options:

- **Title Bar** This option determines whether the title bar is displayed. The title bar contains the title of the folder, view selection options, and the toolbar icons.
- **Folder Picker** This option determines whether to display the folder tree, with which users can navigate to other folders.

Note Regardless of the option(s) chosen, the list of messages is displayed in all cases.

Standard Frameset Options Group Box

The options in the **Standard Frameset Options** group box let you determine how many frames are generated – the original frameset only, or optional second or third framesets – and the size and position of each. **Outlook Web Access Frame Location** can be set to **Top** or **Bottom** if the **Second Frame** or **Third Frame** options are selected; this setting determines the position of the frame that displays folder content. If the **Second Frame** and **Third Frame** options are not selected, the location for the folder-content frame is automatically set to **Full Page**.

If you select the **Second Frame** or the **Third Frame** option, you need to enter the complete location of the target frames. You can also adjust the size of each frame. Do not set other Outlook Web Access frames as targets of the second and third frames, as this would result in unpredictable results. Also, you should verify any target frames before setting these properties.

Caution If you change settings on this property sheet after having edited the ASP file, the changes you made in the ASP file will be overwritten when you close the property sheet.

Error Codes

When the CDO Library or the CDO Rendering Library calls MAPI, the desired return value 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 CDO libraries. 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 Microsoft® Visual Basic® programmer, the CDO libraries define 32-bit type library constants for all relevant warning and error codes. These are provided here in alphabetic and then in numeric order.

A program running on a 16-bit platform cannot use these type library constants. Such a program must test against the low-order word of the constant's value incremented by decimal 1000. For more information on error checking, see [Handling Errors](#).

Note MAPI returns 32-bit values to Visual Basic for both warnings and errors, but Visual Basic treats the two cases differently. Errors are passed to the CDO libraries as 32-bit codes, while warnings are returned as the sum of decimal 1000 and the low-order word of the warning, even on a 32-bit platform. This means that a partial completion, for example, is returned as &H0680 + 1000, or 2664, instead of &H0040680, the full value of **CdoW_PARTIAL_COMPLETION**.

To test for a MAPI warning in a Visual Basic program, you can code the decimal value from the following tables directly into your program. Alternatively, if you prefer to use the type constant for improved readability, you can prepare it for comparison by subtracting the &H40000 bit and then adding 1000 decimal. This is equivalent to subtracting decimal 261144 from the constant:

```
If Err() = CdoW_PARTIAL_COMPLETION - 261144 Then ...
```

Microsoft Visual Basic Scripting Edition (VBScript) and Microsoft® JScript™ do not support any predefined constants. If your application is running as server-side or client-side script, you must use the appropriate hexadecimal or decimal values instead of these type library constants.

The following table lists the return values from MAPI in alphabetic order:

| Warning or error code value
(error code constants available only in
32-bit type libraries) | HRESULT
[VB4 error
value]
(hexadecim
al) | Low-order
word
+ 1000
(decimal) |
|---|---|--|
| CdoE_ACCOUNT_DISABLED | &H8004012
4 | 1292 |
| CdoE_AMBIGUOUS_RECIP | &H8004070
0 | 2792 |
| CdoE_BAD_CHARWIDTH | &H8004010
3 | 1259 |
| CdoE_BAD_COLUMN | &H8004011
8 | 1280 |
| CdoE_BAD_VALUE | &H8004030
1 | 1769 |
| CdoE_BUSY | &H8004010
B | 1267 |
| CdoE_CALL_FAILED | &H8000400 | 17389 |

| | | |
|---|------------|-------|
| | 5 | |
| CdoE_CANCEL | &H8004050 | 2281 |
| | 1 | |
| CdoE_COLLISION | &H8004060 | 2540 |
| | 4 | |
| CdoE_COMPUTED | &H8004011 | 1282 |
| | A | |
| CdoE_CORRUPT_DATA | &H8004011 | 1283 |
| | B | |
| CdoE_CORRUPT_STORE | &H8004060 | 2536 |
| | 0 | |
| CdoE_DECLINE_COPY | &H8004030 | 1774 |
| | 6 | |
| CdoE_DISK_ERROR | &H8004011 | 1278 |
| | 6 | |
| CdoE_END_OF_SESSION | &H8004020 | 1512 |
| | 0 | |
| CdoE_EXTENDED_ERROR | &H8004011 | 1281 |
| | 9 | |
| CdoE_FAILONEPROVIDER | &H8004011 | 1285 |
| | D | |
| CdoE_FOLDER_CYCLE | &H8004060 | 2547 |
| | B | |
| CdoE_HAS_FOLDERS | &H8004060 | 2545 |
| | 9 | |
| CdoE_HAS_MESSAGES | &H8004060 | 2546 |
| | A | |
| CdoE_INTERFACE_NOT_SUPPORTED | &H8000400 | 17386 |
| | 2 | |
| CdoE_INVALID_ACCESS_TIME | &H8004012 | 1291 |
| | 3 | |
| CdoE_INVALID_BOOKMARK | &H8004040 | 2029 |
| | 5 | |
| CdoE_INVALID_ENTRYID | &H8004010 | 1263 |
| | 7 | |
| CdoE_INVALID_OBJECT | &H8004010 | 1264 |
| | 8 | |
| CdoE_INVALID_PARAMETER | &H8007005 | 1087 |
| | 7 | |
| CdoE_INVALID_TYPE | &H8004030 | 1770 |
| | 2 | |
| CdoE_INVALID_WORKSTATION_ACCOUNT | &H8004012 | 1290 |
| | 2 | |
| CdoE_LOGON_FAILED | &H80040111 | 1273 |
| CdoE_MISSING_REQUIRED_COLUMN | &H8004020 | 1514 |
| | 2 | |
| CdoE_NETWORK_ERROR | &H8004011 | 1277 |
| | 5 | |
| CdoE_NO_ACCESS | &H8007000 | 1005 |

| | | |
|--------------------------------------|-----------|------|
| | 5 | |
| CdoE_NO_RECIPIENTS | &H8004060 | 2543 |
| | 7 | |
| CdoE_NO_SUPPORT | &H8004010 | 1258 |
| | 2 | |
| CdoE_NO_SUPPRESS | &H8004060 | 2538 |
| | 2 | |
| CdoE_NON_STANDARD | &H8004060 | 2542 |
| | 6 | |
| CdoE_NOT_ENOUGH_DISK | &H8004010 | 1269 |
| | D | |
| CdoE_NOT_ENOUGH_MEMORY | &H8007000 | 1014 |
| | E | |
| CdoE_NOT_ENOUGH_RESOURCES | &H8004010 | 1270 |
| | E | |
| CdoE_NOT_FOUND | &H8004010 | 1271 |
| | F | |
| CdoE_NOT_IN_QUEUE | &H8004060 | 2537 |
| | 1 | |
| CdoE_NOT_INITIALIZED | &H8004060 | 2541 |
| | 5 | |
| CdoE_NOT_ME | &H8004050 | 2282 |
| | 2 | |
| CdoE_OBJECT_CHANGED | &H8004010 | 1265 |
| | 9 | |
| CdoE_OBJECT_DELETED | &H8004010 | 1266 |
| | A | |
| CdoE_PASSWORD_CHANGE_REQUIRED | &H8004012 | 1288 |
| | 0 | |
| CdoE_PASSWORD_EXPIRED | &H8004012 | 1289 |
| | 1 | |
| CdoE_SESSION_LIMIT | &H8004011 | 1274 |
| | 2 | |
| CdoE_STRING_TOO_LONG | &H8004010 | 1261 |
| | 5 | |
| CdoE_SUBMITTED | &H8004060 | 2544 |
| | 8 | |
| CdoE_TABLE_EMPTY | &H8004040 | 2026 |
| | 2 | |
| CdoE_TABLE_TOO_BIG | &H8004040 | 2027 |
| | 3 | |
| CdoE_TIMEOUT | &H8004040 | 2025 |
| | 1 | |
| CdoE_TOO_BIG | &H8004030 | 1773 |
| | 5 | |
| CdoE_TOO_COMPLEX | &H8004011 | 1279 |
| | 7 | |
| CdoE_TYPE_NO_SUPPORT | &H8004030 | 1771 |
| | 3 | |

| | | |
|--------------------------------|----------------|------|
| CdoE_UNABLE_TO_ABORT | &H8004011
4 | 1276 |
| CdoE_UNABLE_TO_COMPLETE | &H8004040
0 | 2024 |
| CdoE_UNCONFIGURED | &H8004011
C | 1284 |
| CdoE_UNEXPECTED_ID | &H8004030
7 | 1775 |
| CdoE_UNEXPECTED_TYPE | &H8004030
4 | 1772 |
| CdoE_UNKNOWN_CPID | &H8004011
E | 1286 |
| CdoE_UNKNOWN_ENTRYID | &H8004020
1 | 1513 |
| CdoE_UNKNOWN_FLAGS | &H8004010
6 | 1262 |
| CdoE_UNKNOWN_LCID | &H8004011
F | 1287 |
| CdoE_USER_CANCEL | &H8004011
3 | 1275 |
| CdoE_VERSION | &H8004011
0 | 1272 |
| CdoE_WAIT | &H8004050
0 | 2280 |
| CdoW_APPROX_COUNT | &H0004048
2 | 2154 |
| CdoW_CANCEL_MESSAGE | &H0004058
0 | 2408 |
| CdoW_ERRORS_RETURNED | &H0004038
0 | 1896 |
| CdoW_NO_SERVICE | &H0004020
3 | 1515 |
| CdoW_PARTIAL_COMPLETION | &H0004068
0 | 2664 |
| CdoW_POSITION_CHANGED | &H0004048
1 | 2153 |

The following table lists the return values from MAPI in numeric order:

| HRESULT
[VB4 error
value]
(hexadecim
al) | Low-order
word
+ 1000
(decimal) | Warning or error code value

(error code constants available only in
32-bit type libraries) |
|---|--|--|
| &H0004020
3 | 1515 | CdoW_NO_SERVICE |
| &H0004038
0 | 1896 | CdoW_ERRORS_RETURNED |
| &H0004048
1 | 2153 | CdoW_POSITION_CHANGED |

| | | |
|----------------|-------|-------------------------------------|
| &H0004048
2 | 2154 | CdoW_APPROX_COUNT |
| &H0004058
0 | 2408 | CdoW_CANCEL_MESSAGE |
| &H0004068
0 | 2664 | CdoW_PARTIAL_COMPLETION |
| &H800400
2 | 17386 | CdoE_INTERFACE_NOT_SUPPORTED |
| &H800400
5 | 17389 | CdoE_CALL_FAILED |
| &H8004010
2 | 1258 | CdoE_NO_SUPPORT |
| &H8004010
3 | 1259 | CdoE_BAD_CHARWIDTH |
| &H8004010
5 | 1261 | CdoE_STRING_TOO_LONG |
| &H8004010
6 | 1262 | CdoE_UNKNOWN_FLAGS |
| &H8004010
7 | 1263 | CdoE_INVALID_ENTRYID |
| &H8004010
8 | 1264 | CdoE_INVALID_OBJECT |
| &H8004010
9 | 1265 | CdoE_OBJECT_CHANGED |
| &H8004010
A | 1266 | CdoE_OBJECT_DELETED |
| &H8004010
B | 1267 | CdoE_BUSY |
| &H8004010
D | 1269 | CdoE_NOT_ENOUGH_DISK |
| &H8004010
E | 1270 | CdoE_NOT_ENOUGH_RESOURCES |
| &H8004010
F | 1271 | CdoE_NOT_FOUND |
| &H8004011
0 | 1272 | CdoE_VERSION |
| &H8004011 | 1273 | CdoE_LOGON_FAILED |
| &H8004011
2 | 1274 | CdoE_SESSION_LIMIT |
| &H8004011
3 | 1275 | CdoE_USER_CANCEL |
| &H8004011
4 | 1276 | CdoE_UNABLE_TO_ABORT |
| &H8004011
5 | 1277 | CdoE_NETWORK_ERROR |
| &H8004011
6 | 1278 | CdoE_DISK_ERROR |
| &H8004011
7 | 1279 | CdoE_TOO_COMPLEX |

| | | |
|----------------|------|---|
| &H8004011
8 | 1280 | CdoE_BAD_COLUMN |
| &H8004011
9 | 1281 | CdoE_EXTENDED_ERROR |
| &H8004011
A | 1282 | CdoE_COMPUTED |
| &H8004011
B | 1283 | CdoE_CORRUPT_DATA |
| &H8004011
C | 1284 | CdoE_UNCONFIGURED |
| &H8004011
D | 1285 | CdoE_FAILONEPROVIDER |
| &H8004011
E | 1286 | CdoE_UNKNOWN_CPID |
| &H8004011
F | 1287 | CdoE_UNKNOWN_LCID |
| &H8004012
0 | 1288 | CdoE_PASSWORD_CHANGE_REQUIRED |
| &H8004012
1 | 1289 | CdoE_PASSWORD_EXPIRED |
| &H8004012
2 | 1290 | CdoE_INVALID_WORKSTATION_ACCOUNT |
| &H8004012
3 | 1291 | CdoE_INVALID_ACCESS_TIME |
| &H8004012
4 | 1292 | CdoE_ACCOUNT_DISABLED |
| &H8004020
0 | 1512 | CdoE_END_OF_SESSION |
| &H8004020
1 | 1513 | CdoE_UNKNOWN_ENTRYID |
| &H8004020
2 | 1514 | CdoE_MISSING_REQUIRED_COLUMN |
| &H8004030
1 | 1769 | CdoE_BAD_VALUE |
| &H8004030
2 | 1770 | CdoE_INVALID_TYPE |
| &H8004030
3 | 1771 | CdoE_TYPE_NO_SUPPORT |
| &H8004030
4 | 1772 | CdoE_UNEXPECTED_TYPE |
| &H8004030
5 | 1773 | CdoE_TOO_BIG |
| &H8004030
6 | 1774 | CdoE_DECLINE_COPY |
| &H8004030
7 | 1775 | CdoE_UNEXPECTED_ID |
| &H8004040
0 | 2024 | CdoE_UNABLE_TO_COMPLETE |
| &H8004040 | 2025 | CdoE_TIMEOUT |

| | | | |
|-----------|------|--|-------------------------------|
| 1 | | | |
| &H8004040 | 2026 | | CdoE_TABLE_EMPTY |
| 2 | | | |
| &H8004040 | 2027 | | CdoE_TABLE_TOO_BIG |
| 3 | | | |
| &H8004040 | 2029 | | CdoE_INVALID_BOOKMARK |
| 5 | | | |
| &H8004050 | 2280 | | CdoE_WAIT |
| 0 | | | |
| &H8004050 | 2281 | | CdoE_CANCEL |
| 1 | | | |
| &H8004050 | 2282 | | CdoE_NOT_ME |
| 2 | | | |
| &H8004060 | 2536 | | CdoE_CORRUPT_STORE |
| 0 | | | |
| &H8004060 | 2537 | | CdoE_NOT_IN_QUEUE |
| 1 | | | |
| &H8004060 | 2538 | | CdoE_NO_SUPPRESS |
| 2 | | | |
| &H8004060 | 2540 | | CdoE_COLLISION |
| 4 | | | |
| &H8004060 | 2541 | | CdoE_NOT_INITIALIZED |
| 5 | | | |
| &H8004060 | 2542 | | CdoE_NON_STANDARD |
| 6 | | | |
| &H8004060 | 2543 | | CdoE_NO_RECIPIENTS |
| 7 | | | |
| &H8004060 | 2544 | | CdoE_SUBMITTED |
| 8 | | | |
| &H8004060 | 2545 | | CdoE_HAS_FOLDERS |
| 9 | | | |
| &H8004060 | 2546 | | CdoE_HAS_MESSAGES |
| A | | | |
| &H8004060 | 2547 | | CdoE_FOLDER_CYCLE |
| B | | | |
| &H8004070 | 2792 | | CdoE_AMBIGUOUS_RECIP |
| 0 | | | |
| &H8007000 | 1005 | | CdoE_NO_ACCESS |
| 5 | | | |
| &H8007000 | 1014 | | CdoE_NOT_ENOUGH_MEMORY |
| E | | | |
| &H8007005 | 1087 | | CdoE_INVALID_PARAMETER |
| 7 | | | |

MAPI Property Tags

For the convenience of the Microsoft® Visual Basic® programmer, the CDO libraries define 32-bit type library constants for all predefined MAPI properties. These are provided here in alphabetic and then in numeric order.

The Microsoft® Exchange Server and Client define another set of properties in addition to the MAPI properties. For more information, see [Microsoft Exchange Property Tags](#).

Most of the string properties can also be used in the Unicode format. When you reference a property for Unicode, you should change its property type from &H001E to &H001F. For example, the standard property tag for PR_SUBJECT is &H0037001E, but if your application is using Unicode you should refer to PR_SUBJECT with &H0037001F.

Microsoft Visual Basic Scripting Edition (VBScript) and Microsoft® JScript™ do not support any predefined constants. If your application is running as server-side or client-side script, you must use the appropriate hexadecimal or decimal values instead of these type library constants.

For more information on the predefined MAPI properties, see "Properties" and "MAPI Properties" in the *MAPI Programmer's Reference*.

The following table lists the MAPI property tags in alphabetic order:

| Property tag value
(constants available only in 32-bit type libraries) | Hexadecimal
value |
|---|------------------------------|
| CdoPR_7BIT_DISPLAY_NAME | &H39FF001E |
| CdoPR_AB_DEFAULT_DIR | &H3D060102 |
| CdoPR_AB_DEFAULT_PAB | &H3D070102 |
| CdoPR_AB_PROVIDER_ID | &H36150102 |
| CdoPR_AB_PROVIDERS | &H3D010102 |
| CdoPR_AB_SEARCH_PATH | &H3D051102 |
| CdoPR_AB_SEARCH_PATH_UPDATE | &H3D110102 |
| CdoPR_ACCESS | &H0FF40003 |
| CdoPR_ACCESS_LEVEL | &H0FF70003 |
| CdoPR_ACCOUNT | &H3A00001E |
| CdoPR_ACKNOWLEDGEMENT_MODE | &H00010003 |
| CdoPR_ADDRTYPE | &H3002001E |
| CdoPR_ALTERNATE_RECIPIENT | &H3A010102 |

| | |
|--|----------------|
| CdoPR_ALTERNATE_RECIPIENT_ALLOWED | &H0002000
B |
| CdoPR_ANR | &H360C001
E |
| CdoPR_ASSISTANT | &H3A30001
E |
| CdoPR_ASSISTANT_TELEPHONE_NUMBER | &H3A2E001
E |
| CdoPR_ASSOC_CONTENT_COUNT | &H3617000
3 |
| CdoPR_ATTACH_ADDITIONAL_INFO | &H370F010
2 |
| CdoPR_ATTACH_DATA_BIN | &H3701010
2 |
| CdoPR_ATTACH_DATA_OBJ | &H3701000
D |
| CdoPR_ATTACH_ENCODING | &H3702010
2 |
| CdoPR_ATTACH_EXTENSION | &H3703001
E |
| CdoPR_ATTACH_FILENAME | &H3704001
E |
| CdoPR_ATTACH_LONG_FILENAME | &H3707001
E |
| CdoPR_ATTACH_LONG_PATHNAME | &H370D001
E |
| CdoPR_ATTACH_METHOD | &H3705000
3 |
| CdoPR_ATTACH_MIME_TAG | &H370E001
E |
| CdoPR_ATTACH_NUM | &H0E21000
3 |
| CdoPR_ATTACH_PATHNAME | &H3708001
E |
| CdoPR_ATTACH_RENDERING | &H3709010
2 |
| CdoPR_ATTACH_SIZE | &H0E20000
3 |
| CdoPR_ATTACH_TAG | &H370A010
2 |
| CdoPR_ATTACH_TRANSPORT_NAME | &H370C001
E |
| CdoPR_ATTACHMENT_X400_PARAMETERS | &H3700010
2 |
| CdoPR_AUTHORIZING_USERS | &H0003010
2 |
| CdoPR_AUTO_FORWARD_COMMENT | &H0004001
E |
| CdoPR_AUTO_FORWARDED | &H0005000 |

| | |
|--|---------------------|
| CdoPR_BEEPER_TELEPHONE_NUMBER | B
&H3A21001
E |
| CdoPR_BIRTHDAY | &H3A42004
0 |
| CdoPR_BODY | &H1000001
E |
| CdoPR_BODY_CRC | &H0E1C000
3 |
| CdoPR_BUSINESS_ADDRESS_CITY | &H3A27001
E |
| CdoPR_BUSINESS_ADDRESS_COUNTRY | &H3A26001
E |
| CdoPR_BUSINESS_ADDRESS_POST_OFFICE_BOX | &H3A2B001
E |
| CdoPR_BUSINESS_ADDRESS_POSTAL_CODE | &H3A2A001
E |
| CdoPR_BUSINESS_ADDRESS_STATE_OR_PROVINCE | &H3A28001
E |
| CdoPR_BUSINESS_ADDRESS_STREET | &H3A29001
E |
| CdoPR_BUSINESS_FAX_NUMBER | &H3A24001
E |
| CdoPR_BUSINESS_HOME_PAGE | &H3A51001
E |
| CdoPR_BUSINESS_TELEPHONE_NUMBER | &H3A08001
E |
| CdoPR_BUSINESS2_TELEPHONE_NUMBER | &H3A1B001
E |
| CdoPR_CALLBACK_TELEPHONE_NUMBER | &H3A02001
E |
| CdoPR_CAR_TELEPHONE_NUMBER | &H3A1E001
E |
| CdoPR_CELLULAR_TELEPHONE_NUMBER | &H3A1C001
E |
| CdoPR_CHILDRENS_NAMES | &H3A58101
E |
| CdoPR_CLIENT_SUBMIT_TIME | &H0039004
0 |
| CdoPR_COMMENT | &H3004001
E |
| CdoPR_COMMON_VIEWS_ENTRYID | &H35E6010
2 |
| CdoPR_COMPANY_MAIN_PHONE_NUMBER | &H3A57001
E |
| CdoPR_COMPANY_NAME | &H3A16001
E |
| CdoPR_COMPUTER_NETWORK_NAME | &H3A49001
E |

| | |
|---|----------------|
| CdoPR_CONTACT_ADDRTYPES | &H3A54101
E |
| CdoPR_CONTACT_DEFAULT_ADDRESS_INDEX | &H3A55000
3 |
| CdoPR_CONTACT_EMAIL_ADDRESSES | &H3A56101
E |
| CdoPR_CONTACT_ENTRYIDS | &H3A53110
2 |
| CdoPR_CONTACT_VERSION | &H3A52004
8 |
| CdoPR_CONTAINER_CLASS | &H3613001
E |
| CdoPR_CONTAINER_CONTENTS | &H360F000
D |
| CdoPR_CONTAINER_FLAGS | &H3600000
3 |
| CdoPR_CONTAINER_HIERARCHY | &H360E000
D |
| CdoPR_CONTAINER_MODIFY_VERSION | &H3614001
4 |
| CdoPR_CONTENT_CONFIDENTIALITY_ALGORITHM_ID | &H0006010
2 |
| CdoPR_CONTENT_CORRELATOR | &H0007010
2 |
| CdoPR_CONTENT_COUNT | &H3602000
3 |
| CdoPR_CONTENT_IDENTIFIER | &H0008001
E |
| CdoPR_CONTENT_INTEGRITY_CHECK | &H0C00010
2 |
| CdoPR_CONTENT_LENGTH | &H0009000
3 |
| CdoPR_CONTENT_RETURN_REQUESTED | &H000A000
B |
| CdoPR_CONTENT_UNREAD | &H3603000
3 |
| CdoPR_CONTENTS_SORT_ORDER | &H360D100
3 |
| CdoPR_CONTROL_FLAGS | &H3F00000
3 |
| CdoPR_CONTROL_ID | &H3F07010
2 |
| CdoPR_CONTROL_STRUCTURE | &H3F01010
2 |
| CdoPR_CONTROL_TYPE | &H3F02000
3 |
| CdoPR_CONVERSATION_INDEX | &H0071010
2 |
| CdoPR_CONVERSATION_KEY | &H000B010 |

| | |
|---------------------------------------|---------------------|
| CdoPR_CONVERSATION_TOPIC | 2
&H0070001
E |
| CdoPR_CONVERSION_EITS | &H000C010
2 |
| CdoPR_CONVERSION_PROHIBITED | &H3A03000
B |
| CdoPR_CONVERSION_WITH_LOSS_PROHIBITED | &H000D000
B |
| CdoPR_CONVERTED_EITS | &H000E010
2 |
| CdoPR_CORRELATE | &H0E0C000
B |
| CdoPR_CORRELATE_MTSID | &H0E0D010
2 |
| CdoPR_COUNTRY | &H3A26001
E |
| CdoPR_CREATE_TEMPLATES | &H3604000
D |
| CdoPR_CREATION_TIME | &H3007004
0 |
| CdoPR_CREATION_VERSION | &H0E19001
4 |
| CdoPR_CURRENT_VERSION | &H0E00001
4 |
| CdoPR_CUSTOMER_ID | &H3A4A001
E |
| CdoPR_DEF_CREATE_DL | &H3611010
2 |
| CdoPR_DEF_CREATE_MAILUSER | &H3612010
2 |
| CdoPR_DEFAULT_PROFILE | &H3D04000
B |
| CdoPR_DEFAULT_STORE | &H3400000
B |
| CdoPR_DEFAULT_VIEW_ENTRYID | &H3616010
2 |
| CdoPR_DEFERRED_DELIVERY_TIME | &H000F004
0 |
| CdoPR_DELEGATION | &H007E010
2 |
| CdoPR_DELETE_AFTER_SUBMIT | &H0E01000
B |
| CdoPR_DELIVER_TIME | &H0010004
0 |
| CdoPR_DELIVERY_POINT | &H0C07000
3 |
| CdoPR_DELTAX | &H3F03000
3 |

| | |
|---|----------------|
| CdoPR_DELTAY | &H3F04000
3 |
| CdoPR_DEPARTMENT_NAME | &H3A18001
E |
| CdoPR_DEPTH | &H3005000
3 |
| CdoPR_DETAILS_TABLE | &H3605000
D |
| CdoPR_DISC_VAL | &H004A000
B |
| CdoPR_DISCARD_REASON | &H0011000
3 |
| CdoPR_DISCLOSE_RECIPIENTS | &H3A04000
B |
| CdoPR_DISCLOSURE_OF_RECIPIENTS | &H0012000
B |
| CdoPR_DISCRETE_VALUES | &H0E0E000
B |
| CdoPR_DISPLAY_BCC | &H0E02001
E |
| CdoPR_DISPLAY_CC | &H0E03001
E |
| CdoPR_DISPLAY_NAME | &H3001001
E |
| CdoPR_DISPLAY_NAME_PREFIX | &H3A45001
E |
| CdoPR_DISPLAY_TO | &H0E04001
E |
| CdoPR_DISPLAY_TYPE | &H3900000
3 |
| CdoPR_DL_EXPANSION_HISTORY | &H0013010
2 |
| CdoPR_DL_EXPANSION_PROHIBITED | &H0014000
B |
| CdoPR_EMAIL_ADDRESS | &H3003001
E |
| CdoPR_END_DATE | &H0061004
0 |
| CdoPR_ENTRYID | &H0FFF010
2 |
| CdoPR_EXPIRY_TIME | &H0015004
0 |
| CdoPR_EXPLICIT_CONVERSION | &H0C01000
3 |
| CdoPR_FILTERING_HOOKS | &H3D08010
2 |
| CdoPR_FINDER_ENTRYID | &H35E7010
2 |
| CdoPR_FOLDER_ASSOCIATED_CONTENTS | &H3610000 |

| | |
|---|----------------|
| | D |
| CdoPR_FOLDER_TYPE | &H3601000
3 |
| CdoPR_FORM_CATEGORY | &H3304001
E |
| CdoPR_FORM_CATEGORY_SUB | &H3305001
E |
| CdoPR_FORM_CLSID | &H3302004
8 |
| CdoPR_FORM_CONTACT_NAME | &H3303001
E |
| CdoPR_FORM_DESIGNER_GUID | &H3309004
8 |
| CdoPR_FORM_DESIGNER_NAME | &H3308001
E |
| CdoPR_FORM_HIDDEN | &H3307000
B |
| CdoPR_FORM_HOST_MAP | &H3306100
3 |
| CdoPR_FORM_MESSAGE_BEHAVIOR | &H330A000
3 |
| CdoPR_FORM_VERSION | &H3301001
E |
| CdoPR_FTP_SITE | &H3A4C001
E |
| CdoPR_GENDER | &H3A4D000
2 |
| CdoPR_GENERATION | &H3A05001
E |
| CdoPR_GIVEN_NAME | &H3A06001
E |
| CdoPR_GOVERNMENT_ID_NUMBER | &H3A07001
E |
| CdoPR_HASATTACH | &H0E1B000
B |
| CdoPR_HEADER_FOLDER_ENTRYID | &H3E0A010
2 |
| CdoPR_HOBBIES | &H3A43001
E |
| CdoPR_HOME_ADDRESS_CITY | &H3A59001
E |
| CdoPR_HOME_ADDRESS_COUNTRY | &H3A5A001
E |
| CdoPR_HOME_ADDRESS_POST_OFFICE_BOX | &H3A5E001
E |
| CdoPR_HOME_ADDRESS_POSTAL_CODE | &H3A5B001
E |
| CdoPR_HOME_ADDRESS_STATE_OR_PROVINCE | &H3A5C001
E |

| | |
|---|----------------|
| CdoPR_HOME_ADDRESS_STREET | &H3A5D001
E |
| CdoPR_HOME_FAX_NUMBER | &H3A25001
E |
| CdoPR_HOME_TELEPHONE_NUMBER | &H3A09001
E |
| CdoPR_HOME2_TELEPHONE_NUMBER | &H3A2F001
E |
| CdoPR_ICON | &H0FFD010
2 |
| CdoPR_IDENTITY_DISPLAY | &H3E00001
E |
| CdoPR_IDENTITY_ENTRYID | &H3E01010
2 |
| CdoPR_IDENTITY_SEARCH_KEY | &H3E05010
2 |
| CdoPR_IMPLICIT_CONVERSION_PROHIBITED | &H0016000
B |
| CdoPR_IMPORTANCE | &H0017000
3 |
| CdoPR_INCOMPLETE_COPY | &H0035000
B |
| CdoPR_INITIAL_DETAILS_PANE | &H3F08000
3 |
| CdoPR_INITIALS | &H3A0A001
E |
| CdoPR_INSTANCE_KEY | &H0FF6010
2 |
| CdoPR_INTERNET_APPROVED | &H1030001
E |
| CdoPR_INTERNET_ARTICLE_NUMBER | &H0E23000
3 |
| CdoPR_INTERNET_CONTROL | &H1031001
E |
| CdoPR_INTERNET_DISTRIBUTION | &H1032001
E |
| CdoPR_INTERNET_FOLLOWUP_TO | &H1033001
E |
| CdoPR_INTERNET_LINES | &H1034000
3 |
| CdoPR_INTERNET_MESSAGE_ID | &H1035001
E |
| CdoPR_INTERNET_NEWSGROUPS | &H1036001
E |
| CdoPR_INTERNET_NNTP_PATH | &H1038001
E |
| CdoPR_INTERNET_ORGANIZATION | &H1037001
E |
| CdoPR_INTERNET_PRECEDENCE | &H1041001 |

| | |
|---|-----------|
| | E |
| CdoPR_INTERNET_REFERENCES | &H1039001 |
| | E |
| CdoPR_IPM_ID | &H0018010 |
| | 2 |
| CdoPR_IPM_OUTBOX_ENTRYID | &H35E2010 |
| | 2 |
| CdoPR_IPM_OUTBOX_SEARCH_KEY | &H3411010 |
| | 2 |
| CdoPR_IPM_RETURN_REQUESTED | &H0C02000 |
| | B |
| CdoPR_IPM_SENTMAIL_ENTRYID | &H35E4010 |
| | 2 |
| CdoPR_IPM_SENTMAIL_SEARCH_KEY | &H3413010 |
| | 2 |
| CdoPR_IPM_SUBTREE_ENTRYID | &H35E0010 |
| | 2 |
| CdoPR_IPM_SUBTREE_SEARCH_KEY | &H3410010 |
| | 2 |
| CdoPR_IPM_WASTEBASKET_ENTRYID | &H35E3010 |
| | 2 |
| CdoPR_IPM_WASTEBASKET_SEARCH_KEY | &H3412010 |
| | 2 |
| CdoPR_ISDN_NUMBER | &H3A2D001 |
| | E |
| CdoPR_KEYWORD | &H3A0B001 |
| | E |
| CdoPR_LANGUAGE | &H3A0C001 |
| | E |
| CdoPR_LANGUAGES | &H002F001 |
| | E |
| CdoPR_LAST_MODIFICATION_TIME | &H3008004 |
| | 0 |
| CdoPR_LATEST_DELIVERY_TIME | &H0019004 |
| | 0 |
| CdoPR_LOCALITY | &H3A27001 |
| | E |
| CdoPR_LOCATION | &H3A0D001 |
| | E |
| CdoPR_MAIL_PERMISSION | &H3A0E000 |
| | B |
| CdoPR_MANAGER_NAME | &H3A4E001 |
| | E |
| CdoPR_MAPPING_SIGNATURE | &H0FF8010 |
| | 2 |
| CdoPR_MDB_PROVIDER | &H3414010 |
| | 2 |
| CdoPR_MESSAGE_ATTACHMENTS | &H0E13000 |
| | D |

| | |
|--|----------------|
| CdoPR_MESSAGE_CC_ME | &H0058000
B |
| CdoPR_MESSAGE_CLASS | &H001A001
E |
| CdoPR_MESSAGE_DELIVERY_ID | &H001B010
2 |
| CdoPR_MESSAGE_DELIVERY_TIME | &H0E06004
0 |
| CdoPR_MESSAGE_DOWNLOAD_TIME | &H0E18000
3 |
| CdoPR_MESSAGE_FLAGS | &H0E07000
3 |
| CdoPR_MESSAGE_RECIP_ME | &H0059000
B |
| CdoPR_MESSAGE_RECIPIENTS | &H0E12000
D |
| CdoPR_MESSAGE_SECURITY_LABEL | &H001E010
2 |
| CdoPR_MESSAGE_SIZE | &H0E08000
3 |
| CdoPR_MESSAGE_SUBMISSION_ID | &H0047010
2 |
| CdoPR_MESSAGE_TO_ME | &H0057000
B |
| CdoPR_MESSAGE_TOKEN | &H0C03010
2 |
| CdoPR_MHS_COMMON_NAME | &H3A0F001
E |
| CdoPR_MIDDLE_NAME | &H3A44001
E |
| CdoPR_MINI_ICON | &H0FFC010
2 |
| CdoPR_MOBILE_TELEPHONE_NUMBER | &H3A1C001
E |
| CdoPR_MODIFY_VERSION | &H0E1A001
4 |
| CdoPR_MSG_STATUS | &H0E17000
3 |
| CdoPR_NDR_DIAG_CODE | &H0C05000
3 |
| CdoPR_NDR_REASON_CODE | &H0C04000
3 |
| CdoPR_NEWSGROUP_NAME | &H0E24001
E |
| CdoPR_NICKNAME | &H3A4F001
E |
| CdoPR_NNTP_XREF | &H1040001
E |
| CdoPR_NON_RECEIPT_NOTIFICATION_REQUESTED | &H0C06000 |

| | |
|-------------------------------------|---------------------|
| CdoPR_NON_RECEIPT_REASON | B
&H003E000
3 |
| CdoPR_NORMALIZED_SUBJECT | &H0E1D001
E |
| CdoPR_OBJECT_TYPE | &H0FFE000
3 |
| CdoPR_OBSOLETED_IPMS | &H001F010
2 |
| CdoPR_OFFICE_LOCATION | &H3A19001
E |
| CdoPR_OFFICE_TELEPHONE_NUMBER | &H3A08001
E |
| CdoPR_OFFICE2_TELEPHONE_NUMBER | &H3A1B001
E |
| CdoPR_ORGANIZATIONAL_ID_NUMBER | &H3A10001
E |
| CdoPR_ORIG_MESSAGE_CLASS | &H004B001
E |
| CdoPR_ORIGIN_CHECK | &H0027010
2 |
| CdoPR_ORIGINAL_AUTHOR_ADDRTYPE | &H0079001
E |
| CdoPR_ORIGINAL_AUTHOR_EMAIL_ADDRESS | &H007A001
E |
| CdoPR_ORIGINAL_AUTHOR_ENTRYID | &H004C010
2 |
| CdoPR_ORIGINAL_AUTHOR_NAME | &H004D001
E |
| CdoPR_ORIGINAL_AUTHOR_SEARCH_KEY | &H0056010
2 |
| CdoPR_ORIGINAL_DELIVERY_TIME | &H0055004
0 |
| CdoPR_ORIGINAL_DISPLAY_BCC | &H0072001
E |
| CdoPR_ORIGINAL_DISPLAY_CC | &H0073001
E |
| CdoPR_ORIGINAL_DISPLAY_NAME | &H3A13001
E |
| CdoPR_ORIGINAL_DISPLAY_TO | &H0074001
E |
| CdoPR_ORIGINAL_EITS | &H0021010
2 |
| CdoPR_ORIGINAL_ENTRYID | &H3A12010
2 |
| CdoPR_ORIGINAL_SEARCH_KEY | &H3A14010
2 |
| CdoPR_ORIGINAL_SENDER_ADDRTYPE | &H0066001
E |

| | |
|---|----------------|
| CdoPR_ORIGINAL_SENDER_EMAIL_ADDRESS | &H0067001
E |
| CdoPR_ORIGINAL_SENDER_ENTRYID | &H005B010
2 |
| CdoPR_ORIGINAL_SENDER_NAME | &H005A001
E |
| CdoPR_ORIGINAL_SENDER_SEARCH_KEY | &H005C010
2 |
| CdoPR_ORIGINAL_SENSITIVITY | &H002E000
3 |
| CdoPR_ORIGINAL_SENT_REPRESENTING_ADDRTYPE | &H0068001
E |
| CdoPR_ORIGINAL_SENT_REPRESENTING_EMAIL_ADDRESS | &H0069001
E |
| CdoPR_ORIGINAL_SENT_REPRESENTING_ENTRYID | &H005E010
2 |
| CdoPR_ORIGINAL_SENT_REPRESENTING_NAME | &H005D001
E |
| CdoPR_ORIGINAL_SENT_REPRESENTING_SEARCH_KEY | &H005F010
2 |
| CdoPR_ORIGINAL_SUBJECT | &H0049001
E |
| CdoPR_ORIGINAL_SUBMIT_TIME | &H004E004
0 |
| CdoPR_ORIGINALLY_INTENDED_RECIP_ADDRTYPE | &H007B001
E |
| CdoPR_ORIGINALLY_INTENDED_RECIP_EMAIL_ADDRESS | &H007C001
E |
| CdoPR_ORIGINALLY_INTENDED_RECIP_ENTRYID | &H1012010
2 |
| CdoPR_ORIGINALLY_INTENDED_RECIPIENT_NAME | &H0020010
2 |
| CdoPR_ORIGINATING_MTA_CERTIFICATE | &H0E25010
2 |
| CdoPR_ORIGINATOR_AND_DL_EXPANSION_HISTORY | &H1002010
2 |
| CdoPR_ORIGINATOR_CERTIFICATE | &H0022010
2 |
| CdoPR_ORIGINATOR_DELIVERY_REPORT_REQUESTED | &H0023000
B |
| CdoPR_ORIGINATOR_NON_DELIVERY_REPORT_REQUESTED | &H0C08000
B |
| CdoPR_ORIGINATOR_REQUESTED_ALTERNATE_RECIPIENT | &H0C09010
2 |
| CdoPR_ORIGINATOR_RETURN_ADDRESS | &H0024010
2 |
| CdoPR_OTHER_ADDRESS_CITY | &H3A5F001
E |
| CdoPR_OTHER_ADDRESS_COUNTRY | &H3A60001 |

| | |
|---|-----------|
| | E |
| CdoPR_OTHER_ADDRESS_POST_OFFICE_BOX | &H3A64001 |
| | E |
| CdoPR_OTHER_ADDRESS_POSTAL_CODE | &H3A61001 |
| | E |
| CdoPR_OTHER_ADDRESS_STATE_OR_PROVINCE | &H3A62001 |
| | E |
| CdoPR_OTHER_ADDRESS_STREET | &H3A63001 |
| | E |
| CdoPR_OTHER_TELEPHONE_NUMBER | &H3A1F001 |
| | E |
| CdoPR_OWN_STORE_ENTRYID | &H3E06010 |
| | 2 |
| CdoPR_OWNER_APPT_ID | &H0062000 |
| | 3 |
| CdoPR_PAGER_TELEPHONE_NUMBER | &H3A21001 |
| | E |
| CdoPR_PARENT_DISPLAY | &H0E05001 |
| | E |
| CdoPR_PARENT_ENTRYID | &H0E09010 |
| | 2 |
| CdoPR_PARENT_KEY | &H0025010 |
| | 2 |
| CdoPR_PERSONAL_HOME_PAGE | &H3A50001 |
| | E |
| CdoPR_PHYSICAL_DELIVERY_BUREAU_FAX_DELIVERY | &H0C0A000 |
| | B |
| CdoPR_PHYSICAL_DELIVERY_MODE | &H0C0B000 |
| | 3 |
| CdoPR_PHYSICAL_DELIVERY_REPORT_REQUEST | &H0C0C000 |
| | 3 |
| CdoPR_PHYSICAL_FORWARDING_ADDRESS | &H0C0D010 |
| | 2 |
| CdoPR_PHYSICAL_FORWARDING_ADDRESS_REQUESTED | &H0C0E000 |
| | B |
| CdoPR_PHYSICAL_FORWARDING_PROHIBITED | &H0C0F000 |
| | B |
| CdoPR_PHYSICAL_RENDITION_ATTRIBUTES | &H0C10010 |
| | 2 |
| CdoPR_POST_FOLDER_ENTRIES | &H103B010 |
| | 2 |
| CdoPR_POST_FOLDER_NAMES | &H103C001 |
| | E |
| CdoPR_POST_OFFICE_BOX | &H3A2B001 |
| | E |
| CdoPR_POST_REPLY_DENIED | &H103F010 |
| | 2 |
| CdoPR_POST_REPLY_FOLDER_ENTRIES | &H103D010 |
| | 2 |

| | |
|--|----------------|
| CdoPR_POST_REPLY_FOLDER_NAMES | &H103E001
E |
| CdoPR_POSTAL_ADDRESS | &H3A15001
E |
| CdoPR_POSTAL_CODE | &H3A2A001
E |
| CdoPR_PREFERRED_BY_NAME | &H3A47001
E |
| CdoPR_PREPROCESS | &H0E22000
B |
| CdoPR_PRIMARY_CAPABILITY | &H3904010
2 |
| CdoPR_PRIMARY_FAX_NUMBER | &H3A23001
E |
| CdoPR_PRIMARY_TELEPHONE_NUMBER | &H3A1A001
E |
| CdoPR_PRIORITY | &H0026000
3 |
| CdoPR_PROFESSION | &H3A46001
E |
| CdoPR_PROFILE_NAME | &H3D12001
E |
| CdoPR_PROOF_OF_DELIVERY | &H0C11010
2 |
| CdoPR_PROOF_OF_DELIVERY_REQUESTED | &H0C12000
B |
| CdoPR_PROOF_OF_SUBMISSION | &H0E26010
2 |
| CdoPR_PROOF_OF_SUBMISSION_REQUESTED | &H0028000
B |
| CdoPR_PROVIDER_DISPLAY | &H3006001
E |
| CdoPR_PROVIDER_DLL_NAME | &H300A001
E |
| CdoPR_PROVIDER_ORDINAL | &H300D000
3 |
| CdoPR_PROVIDER_SUBMIT_TIME | &H0048004
0 |
| CdoPR_PROVIDER_UID | &H300C010
2 |
| CdoPR_RADIO_TELEPHONE_NUMBER | &H3A1D001
E |
| CdoPR_RCVD_REPRESENTING_ADDRTYPE | &H0077001
E |
| CdoPR_RCVD_REPRESENTING_EMAIL_ADDRESS | &H0078001
E |
| CdoPR_RCVD_REPRESENTING_ENTRYID | &H0043010
2 |
| CdoPR_RCVD_REPRESENTING_NAME | &H0044001 |

| | |
|---|----------------|
| | E |
| CdoPR_RCVD_REPRESENTING_SEARCH_KEY | &H0052010
2 |
| CdoPR_READ_RECEIPT_ENTRYID | &H0046010
2 |
| CdoPR_READ_RECEIPT_REQUESTED | &H0029000
B |
| CdoPR_READ_RECEIPT_SEARCH_KEY | &H0053010
2 |
| CdoPR_RECEIPT_TIME | &H002A004
0 |
| CdoPR_RECEIVE_FOLDER_SETTINGS | &H3415000
D |
| CdoPR_RECEIVED_BY_ADDRTYPE | &H0075001
E |
| CdoPR_RECEIVED_BY_EMAIL_ADDRESS | &H0076001
E |
| CdoPR_RECEIVED_BY_ENTRYID | &H003F010
2 |
| CdoPR_RECEIVED_BY_NAME | &H0040001
E |
| CdoPR_RECEIVED_BY_SEARCH_KEY | &H0051010
2 |
| CdoPR_RECIPIENT_CERTIFICATE | &H0C13010
2 |
| CdoPR_RECIPIENT_NUMBER_FOR_ADVICE | &H0C14001
E |
| CdoPR_RECIPIENT_REASSIGNMENT_PROHIBITED | &H002B000
B |
| CdoPR_RECIPIENT_STATUS | &H0E15000
3 |
| CdoPR_RECIPIENT_TYPE | &H0C15000
3 |
| CdoPR_RECORD_KEY | &H0FF9010
2 |
| CdoPR_REDIRECTION_HISTORY | &H002C010
2 |
| CdoPRREFERRED_BY_NAME | &H3A47001
E |
| CdoPR_REGISTERED_MAIL_TYPE | &H0C16000
3 |
| CdoPR_RELATED_IPMS | &H002D010
2 |
| CdoPR_REMOTE_PROGRESS | &H3E0B000
3 |
| CdoPR_REMOTE_PROGRESS_TEXT | &H3E0C001
E |
| CdoPR_REMOTE_VALIDATE_OK | &H3E0D000
B |

| | |
|--|----------------|
| CdoPR_RENDERING_POSITION | &H370B000
3 |
| CdoPR_REPLY_RECIPIENT_ENTRIES | &H004F010
2 |
| CdoPR_REPLY_RECIPIENT_NAMES | &H0050001
E |
| CdoPR_REPLY_REQUESTED | &H0C17000
B |
| CdoPR_REPLY_TIME | &H0030004
0 |
| CdoPR_REPORT_ENTRYID | &H0045010
2 |
| CdoPR_REPORT_NAME | &H003A001
E |
| CdoPR_REPORT_SEARCH_KEY | &H0054010
2 |
| CdoPR_REPORT_TAG | &H0031010
2 |
| CdoPR_REPORT_TEXT | &H1001001
E |
| CdoPR_REPORT_TIME | &H0032004
0 |
| CdoPR_REPORTING_DL_NAME | &H1003010
2 |
| CdoPR_REPORTING_MTA_CERTIFICATE | &H1004010
2 |
| CdoPR_REQUESTED_DELIVERY_METHOD | &H0C18000
3 |
| CdoPR_RESOURCE_FLAGS | &H3009000
3 |
| CdoPR_RESOURCE_METHODS | &H3E02000
3 |
| CdoPR_RESOURCE_PATH | &H3E07001
E |
| CdoPR_RESOURCE_TYPE | &H3E03000
3 |
| CdoPR_RESPONSE_REQUESTED | &H0063000
B |
| CdoPR_RESPONSIBILITY | &H0E0F000
B |
| CdoPR_RETURNED_IPM | &H0033000
B |
| CdoPR_ROW_TYPE | &H0FF5000
3 |
| CdoPR_ROWID | &H3000000
3 |
| CdoPR_RTF_COMPRESSED | &H1009010
2 |
| CdoPR_RTF_IN_SYNC | &H0E1F000 |

| | |
|---------------------------------------|-----------|
| | B |
| CdoPR_RTF_SYNC_BODY_COUNT | &H1007000 |
| | 3 |
| CdoPR_RTF_SYNC_BODY_CRC | &H1006000 |
| | 3 |
| CdoPR_RTF_SYNC_BODY_TAG | &H1008001 |
| | E |
| CdoPR_RTF_SYNC_PREFIX_COUNT | &H1010000 |
| | 3 |
| CdoPR_RTF_SYNC_TRAILING_COUNT | &H1011000 |
| | 3 |
| CdoPR_SEARCH | &H3607000 |
| | D |
| CdoPR_SEARCH_KEY | &H300B010 |
| | 2 |
| CdoPR_SECURITY | &H0034000 |
| | 3 |
| CdoPR_SELECTABLE | &H3609000 |
| | B |
| CdoPR_SEND_INTERNET_ENCODING | &H3A71000 |
| | 3 |
| CdoPR_SEND_RICH_INFO | &H3A40000 |
| | B |
| CdoPR_SENDER_ADDRTYPE | &H0C1E001 |
| | E |
| CdoPR_SENDER_EMAIL_ADDRESS | &H0C1F001 |
| | E |
| CdoPR_SENDER_ENTRYID | &H0C19010 |
| | 2 |
| CdoPR_SENDER_NAME | &H0C1A001 |
| | E |
| CdoPR_SENDER_SEARCH_KEY | &H0C1D010 |
| | 2 |
| CdoPR_SENSITIVITY | &H0036000 |
| | 3 |
| CdoPR_SENT_REPRESENTING_ADDRTYPE | &H0064001 |
| | E |
| CdoPR_SENT_REPRESENTING_EMAIL_ADDRESS | &H0065001 |
| | E |
| CdoPR_SENT_REPRESENTING_ENTRYID | &H0041010 |
| | 2 |
| CdoPR_SENT_REPRESENTING_NAME | &H0042001 |
| | E |
| CdoPR_SENT_REPRESENTING_SEARCH_KEY | &H003B010 |
| | 2 |
| CdoPR_SENTMAIL_ENTRYID | &H0E0A010 |
| | 2 |
| CdoPR_SERVICE_DELETE_FILES | &H3D10101 |
| | E |

| | |
|------------------------------------|----------------|
| CdoPR_SERVICE_DLL_NAME | &H3D0A001
E |
| CdoPR_SERVICE_ENTRY_NAME | &H3D0B001
E |
| CdoPR_SERVICE_EXTRA_UIDS | &H3D0D010
2 |
| CdoPR_SERVICE_NAME | &H3D09001
E |
| CdoPR_SERVICE_SUPPORT_FILES | &H3D0F101
E |
| CdoPR_SERVICE_UID | &H3D0C010
2 |
| CdoPR_SERVICES | &H3D0E010
2 |
| CdoPR_SPOOLER_STATUS | &H0E10000
3 |
| CdoPR_SPOUSE_NAME | &H3A48001
E |
| CdoPR_START_DATE | &H0060004
0 |
| CdoPR_STATE_OR_PROVINCE | &H3A28001
E |
| CdoPR_STATUS | &H360B000
3 |
| CdoPR_STATUS_CODE | &H3E04000
3 |
| CdoPR_STATUS_STRING | &H3E08001
E |
| CdoPR_STORE_ENTRYID | &H0FFB010
2 |
| CdoPR_STORE_PROVIDERS | &H3D00010
2 |
| CdoPR_STORE_RECORD_KEY | &H0FFA010
2 |
| CdoPR_STORE_STATE | &H340E000
3 |
| CdoPR_STORE_SUPPORT_MASK | &H340D000
3 |
| CdoPR_STREET_ADDRESS | &H3A29001
E |
| CdoPR_SUBFOLDERS | &H360A000
B |
| CdoPR_SUBJECT | &H0037001
E |
| CdoPR_SUBJECT_IPM | &H0038010
2 |
| CdoPR_SUBJECT_PREFIX | &H003D001
E |
| CdoPR_SUBMIT_FLAGS | &H0E14000 |

| | |
|--|---------------------|
| CdoPR_SUPERSEDES | 3
&H103A001
E |
| CdoPR_SUPPLEMENTARY_INFO | &H0C1B001
E |
| CdoPR_SURNAME | &H3A11001
E |
| CdoPR_TELEX_NUMBER | &H3A2C001
E |
| CdoPR_TEMPLATEID | &H3902010
2 |
| CdoPR_TITLE | &H3A17001
E |
| CdoPR_TNEF_CORRELATION_KEY | &H007F010
2 |
| CdoPR_TRANSMITABLE_DISPLAY_NAME | &H3A20001
E |
| CdoPR_TRANSPORT_KEY | &H0E16000
3 |
| CdoPR_TRANSPORT_MESSAGE_HEADERS | &H007D001
E |
| CdoPR_TRANSPORT_PROVIDERS | &H3D02010
2 |
| CdoPR_TRANSPORT_STATUS | &H0E11000
3 |
| CdoPR_TTYTDD_PHONE_NUMBER | &H3A4B001
E |
| CdoPR_TYPE_OF_MTS_USER | &H0C1C000
3 |
| CdoPR_USER_CERTIFICATE | &H3A22010
2 |
| CdoPR_USER_X509_CERTIFICATE | &H3A70110
2 |
| CdoPR_VALID_FOLDER_MASK | &H35DF000
3 |
| CdoPR_VIEWS_ENTRYID | &H35E5010
2 |
| CdoPR_WEDDING_ANNIVERSARY | &H3A41004
0 |
| CdoPR_X400_CONTENT_TYPE | &H003C010
2 |
| CdoPR_X400_DEFERRED_DELIVERY_CANCEL | &H3E09000
B |
| CdoPR_XPOS | &H3F05000
3 |
| CdoPR_YPOS | &H3F06000
3 |

The following table lists the MAPI property tags in numeric order:

| Hexadecimal value | Property tag value
(constants available only in 32-bit type libraries) |
|--------------------------|---|
| &H0001000
3 | CdoPR_ACKNOWLEDGEMENT_MODE |
| &H0002000
B | CdoPR_ALTERNATE_RECIPIENT_ALLOWED |
| &H0003010
2 | CdoPR_AUTHORIZING_USERS |
| &H0004001
E | CdoPR_AUTO_FORWARD_COMMENT |
| &H0005000
B | CdoPR_AUTO_FORWARDED |
| &H0006010
2 | CdoPR_CONTENT_CONFIDENTIALITY_ALGORITHM_ID |
| &H0007010
2 | CdoPR_CONTENT_CORRELATOR |
| &H0008001
E | CdoPR_CONTENT_IDENTIFIER |
| &H0009000
3 | CdoPR_CONTENT_LENGTH |
| &H000A000
B | CdoPR_CONTENT_RETURN_REQUESTED |
| &H000B010
2 | CdoPR_CONVERSATION_KEY |
| &H000C010
2 | CdoPR_CONVERSION_EITS |
| &H000D000
B | CdoPR_CONVERSION_WITH_LOSS_PROHIBITED |
| &H000E010
2 | CdoPR_CONVERTED_EITS |
| &H000F004
0 | CdoPR_DEFERRED_DELIVERY_TIME |
| &H0010004
0 | CdoPR_DELIVER_TIME |
| &H0011000
3 | CdoPR_DISCARD_REASON |
| &H0012000
B | CdoPR_DISCLOSURE_OF_RECIPIENTS |
| &H0013010
2 | CdoPR_DL_EXPANSION_HISTORY |
| &H0014000
B | CdoPR_DL_EXPANSION_PROHIBITED |
| &H0015004
0 | CdoPR_EXPIRY_TIME |
| &H0016000
B | CdoPR_IMPLICIT_CONVERSION_PROHIBITED |
| &H0017000
3 | CdoPR_IMPORTANCE |

| | |
|-----------|--|
| &H0018010 | CdoPR_IPM_ID |
| 2 | |
| &H0019004 | CdoPR_LATEST_DELIVERY_TIME |
| 0 | |
| &H001A001 | CdoPR_MESSAGE_CLASS |
| E | |
| &H001B010 | CdoPR_MESSAGE_DELIVERY_ID |
| 2 | |
| &H001E010 | CdoPR_MESSAGE_SECURITY_LABEL |
| 2 | |
| &H001F010 | CdoPR_OBSOLETED_IPMS |
| 2 | |
| &H0020010 | CdoPR_ORIGINALLY_INTENDED_RECIPIENT_NAME |
| 2 | |
| &H0021010 | CdoPR_ORIGINAL_EITS |
| 2 | |
| &H0022010 | CdoPR_ORIGINATOR_CERTIFICATE |
| 2 | |
| &H0023000 | CdoPR_ORIGINATOR_DELIVERY_REPORT_REQUESTED |
| B | |
| &H0024010 | CdoPR_ORIGINATOR_RETURN_ADDRESS |
| 2 | |
| &H0025010 | CdoPR_PARENT_KEY |
| 2 | |
| &H0026000 | CdoPR_PRIORITY |
| 3 | |
| &H0027010 | CdoPR_ORIGIN_CHECK |
| 2 | |
| &H0028000 | CdoPR_PROOF_OF_SUBMISSION_REQUESTED |
| B | |
| &H0029000 | CdoPR_READ_RECEIPT_REQUESTED |
| B | |
| &H002A004 | CdoPR_RECEIPT_TIME |
| 0 | |
| &H002B000 | CdoPR_RECIPIENT_REASSIGNMENT_PROHIBITED |
| B | |
| &H002C010 | CdoPR_REDIRECTION_HISTORY |
| 2 | |
| &H002D010 | CdoPR_RELATED_IPMS |
| 2 | |
| &H002E000 | CdoPR_ORIGINAL_SENSITIVITY |
| 3 | |
| &H002F001 | CdoPR_LANGUAGES |
| E | |
| &H0030004 | CdoPR_REPLY_TIME |
| 0 | |
| &H0031010 | CdoPR_REPORT_TAG |
| 2 | |
| &H0032004 | CdoPR_REPORT_TIME |

0
&H0033000 CdoPR_RETURNED_IPM
B
&H0034000 CdoPR_SECURITY
3
&H0035000 CdoPR_INCOMPLETE_COPY
B
&H0036000 CdoPR_SENSITIVITY
3
&H0037001 CdoPR_SUBJECT
E
&H0038010 CdoPR_SUBJECT_IPM
2
&H0039004 CdoPR_CLIENT_SUBMIT_TIME
0
&H003A001 CdoPR_REPORT_NAME
E
&H003B010 CdoPR_SENT_REPRESENTING_SEARCH_KEY
2
&H003C010 CdoPR_X400_CONTENT_TYPE
2
&H003D001 CdoPR_SUBJECT_PREFIX
E
&H003E000 CdoPR_NON_RECEIPT_REASON
3
&H003F010 CdoPR_RECEIVED_BY_ENTRYID
2
&H0040001 CdoPR_RECEIVED_BY_NAME
E
&H0041010 CdoPR_SENT_REPRESENTING_ENTRYID
2
&H0042001 CdoPR_SENT_REPRESENTING_NAME
E
&H0043010 CdoPR_RCVD_REPRESENTING_ENTRYID
2
&H0044001 CdoPR_RCVD_REPRESENTING_NAME
E
&H0045010 CdoPR_REPORT_ENTRYID
2
&H0046010 CdoPR_READ_RECEIPT_ENTRYID
2
&H0047010 CdoPR_MESSAGE_SUBMISSION_ID
2
&H0048004 CdoPR_PROVIDER_SUBMIT_TIME
0
&H0049001 CdoPR_ORIGINAL_SUBJECT
E
&H004A000 CdoPR_DISC_VAL
B

&H004B001 CdoPR_ORIG_MESSAGE_CLASS
E
&H004C010 CdoPR_ORIGINAL_AUTHOR_ENTRYID
2
&H004D001 CdoPR_ORIGINAL_AUTHOR_NAME
E
&H004E004 CdoPR_ORIGINAL_SUBMIT_TIME
0
&H004F010 CdoPR_REPLY_RECIPIENT_ENTRIES
2
&H0050001 CdoPR_REPLY_RECIPIENT_NAMES
E
&H0051010 CdoPR_RECEIVED_BY_SEARCH_KEY
2
&H0052010 CdoPR_RCVD_REPRESENTING_SEARCH_KEY
2
&H0053010 CdoPR_READ_RECEIPT_SEARCH_KEY
2
&H0054010 CdoPR_REPORT_SEARCH_KEY
2
&H0055004 CdoPR_ORIGINAL_DELIVERY_TIME
0
&H0056010 CdoPR_ORIGINAL_AUTHOR_SEARCH_KEY
2
&H0057000 CdoPR_MESSAGE_TO_ME
B
&H0058000 CdoPR_MESSAGE_CC_ME
B
&H0059000 CdoPR_MESSAGE_RECIP_ME
B
&H005A001 CdoPR_ORIGINAL_SENDER_NAME
E
&H005B010 CdoPR_ORIGINAL_SENDER_ENTRYID
2
&H005C010 CdoPR_ORIGINAL_SENDER_SEARCH_KEY
2
&H005D001 CdoPR_ORIGINAL_SENT_REPRESENTING_NAME
E
&H005E010 CdoPR_ORIGINAL_SENT_REPRESENTING_ENTRYID
2
&H005F010 CdoPR_ORIGINAL_SENT_REPRESENTING_SEARCH_KEY
2
&H0060004 CdoPR_START_DATE
0
&H0061004 CdoPR_END_DATE
0
&H0062000 CdoPR_OWNER_APPT_ID
3
&H0063000 CdoPR_RESPONSE_REQUESTED

B
&H0064001 CdoPR_SENT_REPRESENTING_ADDRTYPE
E
&H0065001 CdoPR_SENT_REPRESENTING_EMAIL_ADDRESS
E
&H0066001 CdoPR_ORIGINAL_SENDER_ADDRTYPE
E
&H0067001 CdoPR_ORIGINAL_SENDER_EMAIL_ADDRESS
E
&H0068001 CdoPR_ORIGINAL_SENT_REPRESENTING_ADDRTYPE
E
&H0069001 CdoPR_ORIGINAL_SENT_REPRESENTING_EMAIL_
ADDRESS
E
&H0070001 CdoPR_CONVERSATION_TOPIC
E
&H0071010 CdoPR_CONVERSATION_INDEX
2
&H0072001 CdoPR_ORIGINAL_DISPLAY_BCC
E
&H0073001 CdoPR_ORIGINAL_DISPLAY_CC
E
&H0074001 CdoPR_ORIGINAL_DISPLAY_TO
E
&H0075001 CdoPR_RECEIVED_BY_ADDRTYPE
E
&H0076001 CdoPR_RECEIVED_BY_EMAIL_ADDRESS
E
&H0077001 CdoPR_RCVD_REPRESENTING_ADDRTYPE
E
&H0078001 CdoPR_RCVD_REPRESENTING_EMAIL_ADDRESS
E
&H0079001 CdoPR_ORIGINAL_AUTHOR_ADDRTYPE
E
&H007A001 CdoPR_ORIGINAL_AUTHOR_EMAIL_ADDRESS
E
&H007B001 CdoPR_ORIGINALLY_INTENDED_RECIP_ADDRTYPE
E
&H007C001 CdoPR_ORIGINALLY_INTENDED_RECIP_EMAIL_
ADDRESS
E
&H007D001 CdoPR_TRANSPORT_MESSAGE_HEADERS
E
&H007E010 CdoPR_DELEGATION
2
&H007F010 CdoPR_TNEF_CORRELATION_KEY
2
&H0C00010 CdoPR_CONTENT_INTEGRITY_CHECK
2
&H0C01000 CdoPR_EXPLICIT_CONVERSION
3

&H0C02000 CdoPR_IPM_RETURN_REQUESTED
B
&H0C03010 CdoPR_MESSAGE_TOKEN
2
&H0C04000 CdoPR_NDR_REASON_CODE
3
&H0C05000 CdoPR_NDR_DIAG_CODE
3
&H0C06000 CdoPR_NON_RECEIPT_NOTIFICATION_REQUESTED
B
&H0C07000 CdoPR_DELIVERY_POINT
3
&H0C08000 CdoPR_ORIGINATOR_NON_DELIVERY_REPORT_
B REQUESTED
&H0C09010 CdoPR_ORIGINATOR_REQUESTED_ALTERNATE_
2 RECIPIENT
&H0C0A000 CdoPR_PHYSICAL_DELIVERY_BUREAU_FAX_DELIVERY
B
&H0C0B000 CdoPR_PHYSICAL_DELIVERY_MODE
3
&H0C0C000 CdoPR_PHYSICAL_DELIVERY_REPORT_REQUEST
3
&H0C0D010 CdoPR_PHYSICAL_FORWARDING_ADDRESS
2
&H0C0E000 CdoPR_PHYSICAL_FORWARDING_ADDRESS_
B REQUESTED
&H0C0F000 CdoPR_PHYSICAL_FORWARDING_PROHIBITED
B
&H0C10010 CdoPR_PHYSICAL_RENDITION_ATTRIBUTES
2
&H0C11010 CdoPR_PROOF_OF_DELIVERY
2
&H0C12000 CdoPR_PROOF_OF_DELIVERY_REQUESTED
B
&H0C13010 CdoPR_RECIPIENT_CERTIFICATE
2
&H0C14001 CdoPR_RECIPIENT_NUMBER_FOR_ADVICE
E
&H0C15000 CdoPR_RECIPIENT_TYPE
3
&H0C16000 CdoPR_REGISTERED_MAIL_TYPE
3
&H0C17000 CdoPR_REPLY_REQUESTED
B
&H0C18000 CdoPR_REQUESTED_DELIVERY_METHOD
3
&H0C19010 CdoPR_SENDER_ENTRYID
2
&H0C1A001 CdoPR_SENDER_NAME

E
&H0C1B001 CdoPR_SUPPLEMENTARY_INFO
E
&H0C1C000 CdoPR_TYPE_OF_MTS_USER
3
&H0C1D010 CdoPR_SENDER_SEARCH_KEY
2
&H0C1E001 CdoPR_SENDER_ADDRTYPE
E
&H0C1F001 CdoPR_SENDER_EMAIL_ADDRESS
E
&H0E00001 CdoPR_CURRENT_VERSION
4
&H0E01000 CdoPR_DELETE_AFTER_SUBMIT
B
&H0E02001 CdoPR_DISPLAY_BCC
E
&H0E03001 CdoPR_DISPLAY_CC
E
&H0E04001 CdoPR_DISPLAY_TO
E
&H0E05001 CdoPR_PARENT_DISPLAY
E
&H0E06004 CdoPR_MESSAGE_DELIVERY_TIME
0
&H0E07000 CdoPR_MESSAGE_FLAGS
3
&H0E08000 CdoPR_MESSAGE_SIZE
3
&H0E09010 CdoPR_PARENT_ENTRYID
2
&H0E0A010 CdoPR_SENTMAIL_ENTRYID
2
&H0E0C000 CdoPR_CORRELATE
B
&H0E0D010 CdoPR_CORRELATE_MTSID
2
&H0E0E000 CdoPR_DISCRETE_VALUES
B
&H0E0F000 CdoPR_RESPONSIBILITY
B
&H0E10000 CdoPR_SPOOLER_STATUS
3
&H0E11000 CdoPR_TRANSPORT_STATUS
3
&H0E12000 CdoPR_MESSAGE_RECIPIENTS
D
&H0E13000 CdoPR_MESSAGE_ATTACHMENTS
D

| | |
|-----------|-----------------------------------|
| &H0E14000 | CdoPR_SUBMIT_FLAGS |
| 3 | |
| &H0E15000 | CdoPR_RECIPIENT_STATUS |
| 3 | |
| &H0E16000 | CdoPR_TRANSPORT_KEY |
| 3 | |
| &H0E17000 | CdoPR_MSG_STATUS |
| 3 | |
| &H0E18000 | CdoPR_MESSAGE_DOWNLOAD_TIME |
| 3 | |
| &H0E19001 | CdoPR_CREATION_VERSION |
| 4 | |
| &H0E1A001 | CdoPR_MODIFY_VERSION |
| 4 | |
| &H0E1B000 | CdoPR_HASATTACH |
| B | |
| &H0E1C000 | CdoPR_BODY_CRC |
| 3 | |
| &H0E1D001 | CdoPR_NORMALIZED_SUBJECT |
| E | |
| &H0E1F000 | CdoPR_RTF_IN_SYNC |
| B | |
| &H0E20000 | CdoPR_ATTACH_SIZE |
| 3 | |
| &H0E21000 | CdoPR_ATTACH_NUM |
| 3 | |
| &H0E22000 | CdoPR_PREPROCESS |
| B | |
| &H0E23000 | CdoPR_INTERNET_ARTICLE_NUMBER |
| 3 | |
| &H0E24001 | CdoPR_NEWSGROUP_NAME |
| E | |
| &H0E25010 | CdoPR_ORIGINATING_MTA_CERTIFICATE |
| 2 | |
| &H0E26010 | CdoPR_PROOF_OF_SUBMISSION |
| 2 | |
| &H0FF4000 | CdoPR_ACCESS |
| 3 | |
| &H0FF5000 | CdoPR_ROW_TYPE |
| 3 | |
| &H0FF6010 | CdoPR_INSTANCE_KEY |
| 2 | |
| &H0FF7000 | CdoPR_ACCESS_LEVEL |
| 3 | |
| &H0FF8010 | CdoPR_MAPPING_SIGNATURE |
| 2 | |
| &H0FF9010 | CdoPR_RECORD_KEY |
| 2 | |
| &H0FFA010 | CdoPR_STORE_RECORD_KEY |

2
&H0FFB010 CdoPR_STORE_ENTRYID
2
&H0FFC010 CdoPR_MINI_ICON
2
&H0FFD010 CdoPR_ICON
2
&H0FFE000 CdoPR_OBJECT_TYPE
3
&H0FFF010 CdoPR_ENTRYID
2
&H1000001 CdoPR_BODY
E
&H1001001 CdoPR_REPORT_TEXT
E
&H1002010 CdoPR_ORIGINATOR_AND_DL_EXPANSION_HISTORY
2
&H1003010 CdoPR_REPORTING_DL_NAME
2
&H1004010 CdoPR_REPORTING_MTA_CERTIFICATE
2
&H1006000 CdoPR_RTF_SYNC_BODY_CRC
3
&H1007000 CdoPR_RTF_SYNC_BODY_COUNT
3
&H1008001 CdoPR_RTF_SYNC_BODY_TAG
E
&H1009010 CdoPR_RTF_COMPRESSED
2
&H1010000 CdoPR_RTF_SYNC_PREFIX_COUNT
3
&H1011000 CdoPR_RTF_SYNC_TRAILING_COUNT
3
&H1012010 CdoPR_ORIGINALY_INTENDED_RECIP_ENTRYID
2
&H1030001 CdoPR_INTERNET_APPROVED
E
&H1031001 CdoPR_INTERNET_CONTROL
E
&H1032001 CdoPR_INTERNET_DISTRIBUTION
E
&H1033001 CdoPR_INTERNET_FOLLOWUP_TO
E
&H1034000 CdoPR_INTERNET_LINES
3
&H1035001 CdoPR_INTERNET_MESSAGE_ID
E
&H1036001 CdoPR_INTERNET_NEWSGROUPS
E

&H1037001 CdoPR_INTERNET_ORGANIZATION
E
&H1038001 CdoPR_INTERNET_NNTP_PATH
E
&H1039001 CdoPR_INTERNET_REFERENCES
E
&H103A001 CdoPR_SUPERSEDES
E
&H103B010 CdoPR_POST_FOLDER_ENTRIES
2
&H103C001 CdoPR_POST_FOLDER_NAMES
E
&H103D010 CdoPR_POST_REPLY_FOLDER_ENTRIES
2
&H103E001 CdoPR_POST_REPLY_FOLDER_NAMES
E
&H103F010 CdoPR_POST_REPLY_DENIED
2
&H1040001 CdoPR_NNTP_XREF
E
&H1041001 CdoPR_INTERNET_PRECEDENCE
E
&H3000000 CdoPR_ROWID
3
&H3001001 CdoPR_DISPLAY_NAME
E
&H3002001 CdoPR_ADDRTYPE
E
&H3003001 CdoPR_EMAIL_ADDRESS
E
&H3004001 CdoPR_COMMENT
E
&H3005000 CdoPR_DEPTH
3
&H3006001 CdoPR_PROVIDER_DISPLAY
E
&H3007004 CdoPR_CREATION_TIME
0
&H3008004 CdoPR_LAST_MODIFICATION_TIME
0
&H3009000 CdoPR_RESOURCE_FLAGS
3
&H300A001 CdoPR_PROVIDER_DLL_NAME
E
&H300B010 CdoPR_SEARCH_KEY
2
&H300C010 CdoPR_PROVIDER_UID
2
&H300D000 CdoPR_PROVIDER_ORDINAL

3
&H3301001 CdoPR_FORM_VERSION
E
&H3302004 CdoPR_FORM_CLSID
8
&H3303001 CdoPR_FORM_CONTACT_NAME
E
&H3304001 CdoPR_FORM_CATEGORY
E
&H3305001 CdoPR_FORM_CATEGORY_SUB
E
&H3306100 CdoPR_FORM_HOST_MAP
3
&H3307000 CdoPR_FORM_HIDDEN
B
&H3308001 CdoPR_FORM_DESIGNER_NAME
E
&H3309004 CdoPR_FORM_DESIGNER_GUID
8
&H330A000 CdoPR_FORM_MESSAGE_BEHAVIOR
3
&H3400000 CdoPR_DEFAULT_STORE
B
&H340D000 CdoPR_STORE_SUPPORT_MASK
3
&H340E000 CdoPR_STORE_STATE
3
&H3410010 CdoPR_IPM_SUBTREE_SEARCH_KEY
2
&H3411010 CdoPR_IPM_OUTBOX_SEARCH_KEY
2
&H3412010 CdoPR_IPM_WASTEBASKET_SEARCH_KEY
2
&H3413010 CdoPR_IPM_SENTMAIL_SEARCH_KEY
2
&H3414010 CdoPR_MDB_PROVIDER
2
&H3415000 CdoPR_RECEIVE_FOLDER_SETTINGS
D
&H35DF000 CdoPR_VALID_FOLDER_MASK
3
&H35E0010 CdoPR_IPM_SUBTREE_ENTRYID
2
&H35E2010 CdoPR_IPM_OUTBOX_ENTRYID
2
&H35E3010 CdoPR_IPM_WASTEBASKET_ENTRYID
2
&H35E4010 CdoPR_IPM_SENTMAIL_ENTRYID
2

| | |
|-----------|----------------------------------|
| &H35E5010 | CdoPR_VIEWS_ENTRYID |
| 2 | |
| &H35E6010 | CdoPR_COMMON_VIEWS_ENTRYID |
| 2 | |
| &H35E7010 | CdoPR_FINDER_ENTRYID |
| 2 | |
| &H3600000 | CdoPR_CONTAINER_FLAGS |
| 3 | |
| &H3601000 | CdoPR_FOLDER_TYPE |
| 3 | |
| &H3602000 | CdoPR_CONTENT_COUNT |
| 3 | |
| &H3603000 | CdoPR_CONTENT_UNREAD |
| 3 | |
| &H3604000 | CdoPR_CREATE_TEMPLATES |
| D | |
| &H3605000 | CdoPR_DETAILS_TABLE |
| D | |
| &H3607000 | CdoPR_SEARCH |
| D | |
| &H3609000 | CdoPR_SELECTABLE |
| B | |
| &H360A000 | CdoPR_SUBFOLDERS |
| B | |
| &H360B000 | CdoPR_STATUS |
| 3 | |
| &H360C001 | CdoPR_ANR |
| E | |
| &H360D100 | CdoPR_CONTENTS_SORT_ORDER |
| 3 | |
| &H360E000 | CdoPR_CONTAINER_HIERARCHY |
| D | |
| &H360F000 | CdoPR_CONTAINER_CONTENTS |
| D | |
| &H3610000 | CdoPR_FOLDER_ASSOCIATED_CONTENTS |
| D | |
| &H3611010 | CdoPR_DEF_CREATE_DL |
| 2 | |
| &H3612010 | CdoPR_DEF_CREATE_MAILUSER |
| 2 | |
| &H3613001 | CdoPR_CONTAINER_CLASS |
| E | |
| &H3614001 | CdoPR_CONTAINER_MODIFY_VERSION |
| 4 | |
| &H3615010 | CdoPR_AB_PROVIDER_ID |
| 2 | |
| &H3616010 | CdoPR_DEFAULT_VIEW_ENTRYID |
| 2 | |
| &H3617000 | CdoPR_ASSOC_CONTENT_COUNT |

3
&H3700010 CdoPR_ATTACHMENT_X400_PARAMETERS
2
&H3701000 CdoPR_ATTACH_DATA_OBJ
D
&H3701010 CdoPR_ATTACH_DATA_BIN
2
&H3702010 CdoPR_ATTACH_ENCODING
2
&H3703001 CdoPR_ATTACH_EXTENSION
E
&H3704001 CdoPR_ATTACH_FILENAME
E
&H3705000 CdoPR_ATTACH_METHOD
3
&H3707001 CdoPR_ATTACH_LONG_FILENAME
E
&H3708001 CdoPR_ATTACH_PATHNAME
E
&H3709010 CdoPR_ATTACH_RENDERING
2
&H370A010 CdoPR_ATTACH_TAG
2
&H370B000 CdoPR_RENDERING_POSITION
3
&H370C001 CdoPR_ATTACH_TRANSPORT_NAME
E
&H370D001 CdoPR_ATTACH_LONG_PATHNAME
E
&H370E001 CdoPR_ATTACH_MIME_TAG
E
&H370F010 CdoPR_ATTACH_ADDITIONAL_INFO
2
&H3900000 CdoPR_DISPLAY_TYPE
3
&H3902010 CdoPR_TEMPLATEID
2
&H3904010 CdoPR_PRIMARY_CAPABILITY
2
&H39FF001 CdoPR_7BIT_DISPLAY_NAME
E
&H3A00001 CdoPR_ACCOUNT
E
&H3A01010 CdoPR_ALTERNATE_RECIPIENT
2
&H3A02001 CdoPR_CALLBACK_TELEPHONE_NUMBER
E
&H3A03000 CdoPR_CONVERSION_PROHIBITED
B

&H3A04000 CdoPR_DISCLOSE_RECIPIENTS
B
&H3A05001 CdoPR_GENERATION
E
&H3A06001 CdoPR_GIVEN_NAME
E
&H3A07001 CdoPR_GOVERNMENT_ID_NUMBER
E
&H3A08001 CdoPR_BUSINESS_TELEPHONE_NUMBER
E
&H3A08001 CdoPR_OFFICE_TELEPHONE_NUMBER
E
&H3A09001 CdoPR_HOME_TELEPHONE_NUMBER
E
&H3A0A001 CdoPR_INITIALS
E
&H3A0B001 CdoPR_KEYWORD
E
&H3A0C001 CdoPR_LANGUAGE
E
&H3A0D001 CdoPR_LOCATION
E
&H3A0E000 CdoPR_MAIL_PERMISSION
B
&H3A0F001 CdoPR_MHS_COMMON_NAME
E
&H3A10001 CdoPR_ORGANIZATIONAL_ID_NUMBER
E
&H3A11001 CdoPR_SURNAME
E
&H3A12010 CdoPR_ORIGINAL_ENTRYID
2
&H3A13001 CdoPR_ORIGINAL_DISPLAY_NAME
E
&H3A14010 CdoPR_ORIGINAL_SEARCH_KEY
2
&H3A15001 CdoPR_POSTAL_ADDRESS
E
&H3A16001 CdoPR_COMPANY_NAME
E
&H3A17001 CdoPR_TITLE
E
&H3A18001 CdoPR_DEPARTMENT_NAME
E
&H3A19001 CdoPR_OFFICE_LOCATION
E
&H3A1A001 CdoPR_PRIMARY_TELEPHONE_NUMBER
E
&H3A1B001 CdoPR_BUSINESS2_TELEPHONE_NUMBER

E
&H3A1B001 CdoPR_OFFICE2_TELEPHONE_NUMBER
E
&H3A1C001 CdoPR_CELLULAR_TELEPHONE_NUMBER
E
&H3A1C001 CdoPR_MOBILE_TELEPHONE_NUMBER
E
&H3A1D001 CdoPR_RADIO_TELEPHONE_NUMBER
E
&H3A1E001 CdoPR_CAR_TELEPHONE_NUMBER
E
&H3A1F001 CdoPR_OTHER_TELEPHONE_NUMBER
E
&H3A20001 CdoPR_TRANSMITABLE_DISPLAY_NAME
E
&H3A21001 CdoPR_BEEPER_TELEPHONE_NUMBER
E
&H3A21001 CdoPR_PAGER_TELEPHONE_NUMBER
E
&H3A22010 CdoPR_USER_CERTIFICATE
2
&H3A23001 CdoPR_PRIMARY_FAX_NUMBER
E
&H3A24001 CdoPR_BUSINESS_FAX_NUMBER
E
&H3A25001 CdoPR_HOME_FAX_NUMBER
E
&H3A26001 CdoPR_BUSINESS_ADDRESS_COUNTRY
E
&H3A26001 CdoPR_COUNTRY
E
&H3A27001 CdoPR_BUSINESS_ADDRESS_CITY
E
&H3A27001 CdoPR_LOCALITY
E
&H3A28001 CdoPR_BUSINESS_ADDRESS_STATE_OR_PROVINCE
E
&H3A28001 CdoPR_STATE_OR_PROVINCE
E
&H3A29001 CdoPR_BUSINESS_ADDRESS_STREET
E
&H3A29001 CdoPR_STREET_ADDRESS
E
&H3A2A001 CdoPR_BUSINESS_ADDRESS_POSTAL_CODE
E
&H3A2A001 CdoPR_POSTAL_CODE
E
&H3A2B001 CdoPR_BUSINESS_ADDRESS_POST_OFFICE_BOX
E

&H3A2B001 CdoPR_POST_OFFICE_BOX
E
&H3A2C001 CdoPR_TELEX_NUMBER
E
&H3A2D001 CdoPR_ISDN_NUMBER
E
&H3A2E001 CdoPR_ASSISTANT_TELEPHONE_NUMBER
E
&H3A2F001 CdoPR_HOME2_TELEPHONE_NUMBER
E
&H3A30001 CdoPR_ASSISTANT
E
&H3A40000 CdoPR_SEND_RICH_INFO
B
&H3A41004 CdoPR_WEDDING_ANNIVERSARY
0
&H3A42004 CdoPR_BIRTHDAY
0
&H3A43001 CdoPR_HOBBIES
E
&H3A44001 CdoPR_MIDDLE_NAME
E
&H3A45001 CdoPR_DISPLAY_NAME_PREFIX
E
&H3A46001 CdoPR_PROFESSION
E
&H3A47001 CdoPR_PREFERRED_BY_NAME
E
&H3A47001 CdoPRREFERRED_BY_NAME
E
&H3A48001 CdoPR_SPOUSE_NAME
E
&H3A49001 CdoPR_COMPUTER_NETWORK_NAME
E
&H3A4A001 CdoPR_CUSTOMER_ID
E
&H3A4B001 CdoPR_TTYTDD_PHONE_NUMBER
E
&H3A4C001 CdoPR_FTP_SITE
E
&H3A4D000 CdoPR_GENDER
2
&H3A4E001 CdoPR_MANAGER_NAME
E
&H3A4F001 CdoPR_NICKNAME
E
&H3A50001 CdoPR_PERSONAL_HOME_PAGE
E
&H3A51001 CdoPR_BUSINESS_HOME_PAGE

E
&H3A52004 CdoPR_CONTACT_VERSION
8
&H3A53110 CdoPR_CONTACT_ENTRYIDS
2
&H3A54101 CdoPR_CONTACT_ADDRTYPES
E
&H3A55000 CdoPR_CONTACT_DEFAULT_ADDRESS_INDEX
3
&H3A56101 CdoPR_CONTACT_EMAIL_ADDRESSES
E
&H3A57001 CdoPR_COMPANY_MAIN_PHONE_NUMBER
E
&H3A58101 CdoPR_CHILDRENS_NAMES
E
&H3A59001 CdoPR_HOME_ADDRESS_CITY
E
&H3A5A001 CdoPR_HOME_ADDRESS_COUNTRY
E
&H3A5B001 CdoPR_HOME_ADDRESS_POSTAL_CODE
E
&H3A5C001 CdoPR_HOME_ADDRESS_STATE_OR_PROVINCE
E
&H3A5D001 CdoPR_HOME_ADDRESS_STREET
E
&H3A5E001 CdoPR_HOME_ADDRESS_POST_OFFICE_BOX
E
&H3A5F001 CdoPR_OTHER_ADDRESS_CITY
E
&H3A60001 CdoPR_OTHER_ADDRESS_COUNTRY
E
&H3A61001 CdoPR_OTHER_ADDRESS_POSTAL_CODE
E
&H3A62001 CdoPR_OTHER_ADDRESS_STATE_OR_PROVINCE
E
&H3A63001 CdoPR_OTHER_ADDRESS_STREET
E
&H3A64001 CdoPR_OTHER_ADDRESS_POST_OFFICE_BOX
E
&H3A70110 CdoPR_USER_X509_CERTIFICATE
2
&H3A71000 CdoPR_SEND_INTERNET_ENCODING
3
&H3D00010 CdoPR_STORE_PROVIDERS
2
&H3D01010 CdoPR_AB_PROVIDERS
2
&H3D02010 CdoPR_TRANSPORT_PROVIDERS
2

| | |
|----------------|-------------------------------------|
| &H3D04000
B | CdoPR_DEFAULT_PROFILE |
| &H3D05110
2 | CdoPR_AB_SEARCH_PATH |
| &H3D06010
2 | CdoPR_AB_DEFAULT_DIR |
| &H3D07010
2 | CdoPR_AB_DEFAULT_PAB |
| &H3D08010
2 | CdoPR_FILTERING_HOOKS |
| &H3D09001
E | CdoPR_SERVICE_NAME |
| &H3D0A001
E | CdoPR_SERVICE_DLL_NAME |
| &H3D0B001
E | CdoPR_SERVICE_ENTRY_NAME |
| &H3D0C010
2 | CdoPR_SERVICE_UID |
| &H3D0D010
2 | CdoPR_SERVICE_EXTRA_UIDS |
| &H3D0E010
2 | CdoPR_SERVICES |
| &H3D0F101
E | CdoPR_SERVICE_SUPPORT_FILES |
| &H3D10101
E | CdoPR_SERVICE_DELETE_FILES |
| &H3D11010
2 | CdoPR_AB_SEARCH_PATH_UPDATE |
| &H3D12001
E | CdoPR_PROFILE_NAME |
| &H3E00001
E | CdoPR_IDENTITY_DISPLAY |
| &H3E01010
2 | CdoPR_IDENTITY_ENTRYID |
| &H3E02000
3 | CdoPR_RESOURCE_METHODS |
| &H3E03000
3 | CdoPR_RESOURCE_TYPE |
| &H3E04000
3 | CdoPR_STATUS_CODE |
| &H3E05010
2 | CdoPR_IDENTITY_SEARCH_KEY |
| &H3E06010
2 | CdoPR_OWN_STORE_ENTRYID |
| &H3E07001
E | CdoPR_RESOURCE_PATH |
| &H3E08001
E | CdoPR_STATUS_STRING |
| &H3E09000 | CdoPR_X400_DEFERRED_DELIVERY_CANCEL |

B
&H3E0A010 CdoPR_HEADER_FOLDER_ENTRYID
2
&H3E0B000 CdoPR_REMOTE_PROGRESS
3
&H3E0C001 CdoPR_REMOTE_PROGRESS_TEXT
E
&H3E0D000 CdoPR_REMOTE_VALIDATE_OK
B
&H3F00000 CdoPR_CONTROL_FLAGS
3
&H3F01010 CdoPR_CONTROL_STRUCTURE
2
&H3F02000 CdoPR_CONTROL_TYPE
3
&H3F03000 CdoPR_DELTAX
3
&H3F04000 CdoPR_DELTAY
3
&H3F05000 CdoPR_XPOS
3
&H3F06000 CdoPR_YPOS
3
&H3F07010 CdoPR_CONTROL_ID
2
&H3F08000 CdoPR_INITIAL_DETAILS_PANE
3

Microsoft Exchange Property Tags

Microsoft® Exchange defines a set of properties that are used by the Microsoft Exchange Server and the Microsoft Exchange Client. The Microsoft® Visual Basic® programmer may occasionally need to access some of these properties from a CDO application, so they are provided here in alphabetic and then in numeric order.

The Messaging Application Program Interface (MAPI) defines a set of properties that are independent of Microsoft Exchange. The CDO libraries define 32-bit type library constants for these predefined MAPI properties. For more information, see [MAPI Property Tags](#).

The CDO libraries do not provide any type library constants for the Microsoft Exchange properties. If you need to access one of them, you should assign its hexadecimal or decimal value to a variable:

```
PR_EMS_AB_EXTENSION_ATTRIBUTE_1 = &H802D001E
```

Most of the string properties can also be used in the Unicode format. When you reference a property for Unicode, you should change its property type from &H001E to &H001F. For example, the standard property tag for PR_EMS_AB_EXTENSION_ATTRIBUTE_10 is &H8036001E, but if your application is using Unicode you should refer to this property with &H8036001F.

Note that CDO does not support access to properties of the following types:

- PT_CLSID, where the last four digits of the property tag are 0048
- PT_OBJECT, where the last four digits of the property tag are 000D

For more information on Microsoft Exchange properties, see the *Microsoft Exchange Server Programmer's Reference* in the "Database and Messaging Services" section of the Microsoft® Platform SDK.

The following table lists the Microsoft Exchange property tags in alphabetic order:

| Property tag (not in type library) | Hexadecimal value |
|---|--------------------------|
| PR_ABSTRACT | &H3FDA001E |
| PR_ACL_DATA | &H3FE00102 |
| PR_ACL_TABLE | &H3FE0000D |
| PR_ACTIVE_USER_ENTRYID | &H66520102 |
| PR_ADDRBOOK_FOR_LOCAL_SITE_ENTRYID | &H66260102 |
| PR_ADDRESS_BOOK_DISPLAY_NAME | &H3FE8001E |
| PR_ADDRESS_BOOK_ENTRYID | &H663B0102 |
| PR_ARRIVAL_TIME | &H665F0040 |
| PR_ASSOC_MESSAGE_SIZE | &H66B40003 |
| PR_ASSOC_MESSAGE_SIZE_EXTENDED | &H66B4001 |

| | |
|-------------------------------|-----------|
| | 4 |
| PR_ASSOC_MSG_W_ATTACH_COUNT | &H66AE000 |
| | 3 |
| PR_ATTACH_ON_ASSOC_MSG_COUNT | &H66B2000 |
| | 3 |
| PR_ATTACH_ON_NORMAL_MSG_COUNT | &H66B1000 |
| | 3 |
| PR_AUTO_ADD_NEW_SUBS | &H65E5000 |
| | B |
| PR_AUTO_RESPONSE_SUPPRESS | &H3FDF000 |
| | 3 |
| PR_BILATERAL_INFO | &H3FDC010 |
| | 2 |
| PR_CACHED_COLUMN_COUNT | &H66AC000 |
| | 3 |
| PR_CATEG_COUNT | &H66AB000 |
| | 3 |
| PR_CHANGE_ADVISOR | &H6634000 |
| | D |
| PR_CHANGE_KEY | &H65E2010 |
| | 2 |
| PR_CHANGE_NOTIFICATION_GUID | &H6637004 |
| | 8 |
| PR_CLIENT_ACTIONS | &H6645010 |
| | 2 |
| PR_CODE_PAGE_ID | &H66C3000 |
| | 3 |
| PR_COLLECTOR | &H662E000 |
| | D |
| PR_CONFLICT_ENTRYID | &H3FF0010 |
| | 2 |
| PR_CONTACT_COUNT | &H66B7000 |
| | 3 |
| PR_CONTENT_SEARCH_KEY | &H6666010 |
| | 2 |
| PR_CONTENTS_SYNCHRONIZER | &H662D000 |
| | D |
| PR_CREATOR_ENTRYID | &H3FF9010 |
| | 2 |
| PR_CREATOR_NAME | &H3FF8001 |
| | E |
| PR_DAM_BACK_PATCHED | &H6647000 |
| | B |
| PR_DAM_ORIGINAL_ENTRYID | &H6646010 |
| | 2 |
| PR_DEFERRED_SEND_NUMBER | &H3FEB000 |
| | 3 |
| PR_DEFERRED_SEND_TIME | &H3FEF004 |
| | 0 |

| | |
|---|----------------|
| PR_DEFERRED_SEND_UNITS | &H3FEC000
3 |
| PR_DELEGATED_BY_RULE | &H3FE3000
B |
| PR_DELETED_ASSOC_MESSAGE_SIZE_EXTENDED | &H669D001
4 |
| PR_DELETED_ASSOC_MSG_COUNT | &H6643000
3 |
| PR_DELETED_FOLDER_COUNT | &H6641000
3 |
| PR_DELETED_MESSAGE_SIZE_EXTENDED | &H669B001
4 |
| PR_DELETED_MSG_COUNT | &H6640000
3 |
| PR_DELETED_NORMAL_MESSAGE_SIZE_EXTENDED | &H669C001
4 |
| PR_DELETED_ON | &H668F004
0 |
| PR_DESIGN_IN_PROGRESS | &H3FE4000
B |
| PR_DISABLE_FULL_FIDELITY | &H10F2000
B |
| PR_DISABLE_WINSOCK | &H6618000
3 |
| PR_DL_REPORT_FLAGS | &H3FDB000
3 |
| PR_EFORMS_FOR_LOCALE_ENTRYID | &H6624010
2 |
| PR_EFORMS_LOCALE_ID | &H3FE9000
3 |
| PR_EFORMS_REGISTRY_ENTRYID | &H6621010
2 |
| PR_EMS_AB_ACCESS_CATEGORY | &H8044000
3 |
| PR_EMS_AB_ACTIVATION_SCHEDULE | &H8045010
2 |
| PR_EMS_AB_ACTIVATION_STYLE | &H8046000
3 |
| PR_EMS_AB_ADDRESS_ENTRY_DISPLAY_TABLE | &H8017010
2 |
| PR_EMS_AB_ADDRESS_ENTRY_DISPLAY_TABLE_MSDO
S | &H8047010
2 |
| PR_EMS_AB_ADDRESS_SYNTAX | &H8018010
2 |
| PR_EMS_AB_ADDRESS_TYPE | &H8048001
E |
| PR_EMS_AB_ADMD | &H8049001
E |
| PR_EMS_AB_ADMIN_DESCRIPTION | &H804A001 |

| | |
|-------------------------------------|-----------|
| | E |
| PR_EMS_AB_ADMIN_DISPLAY_NAME | &H804B001 |
| | E |
| PR_EMS_AB_ADMIN_EXTENSION_DLL | &H804C001 |
| | E |
| PR_EMS_AB_ALIASED_OBJECT_NAME | &H804D001 |
| | E |
| PR_EMS_AB_ALIASED_OBJECT_NAME_O | &H804D000 |
| | D |
| PR_EMS_AB_ALT_RECIPIENT | &H804E001 |
| | E |
| PR_EMS_AB_ALT_RECIPIENT_BL | &H804F101 |
| | E |
| PR_EMS_AB_ALT_RECIPIENT_BL_O | &H804F000 |
| | D |
| PR_EMS_AB_ALT_RECIPIENT_O | &H804E000 |
| | D |
| PR_EMS_AB_ANCESTOR_ID | &H8050010 |
| | 2 |
| PR_EMS_AB_ANONYMOUS_ACCESS | &H8187000 |
| | B |
| PR_EMS_AB_ANONYMOUS_ACCOUNT | &H8C26001 |
| | E |
| PR_EMS_AB_ASSOC_NT_ACCOUNT | &H8027010 |
| | 2 |
| PR_EMS_AB_ASSOC_PROTOCOL_CFG_NNTP | &H81A5001 |
| | E |
| PR_EMS_AB_ASSOC_PROTOCOL_CFG_NNTP_O | &H81A5000 |
| | D |
| PR_EMS_AB_ASSOC_REMOTE_DXA | &H8051101 |
| | E |
| PR_EMS_AB_ASSOC_REMOTE_DXA_O | &H8051000 |
| | D |
| PR_EMS_AB_ASSOCIATION_LIFETIME | &H8052000 |
| | 3 |
| PR_EMS_AB_ATTRIBUTE_CERTIFICATE | &H8C45110 |
| | 2 |
| PR_EMS_AB_AUTH_ORIG_BL | &H8053101 |
| | E |
| PR_EMS_AB_AUTH_ORIG_BL_O | &H8053000 |
| | D |
| PR_EMS_AB_AUTHENTICATION_TO_USE | &H819A001 |
| | E |
| PR_EMS_AB_AUTHORITY_REVOCATION_LIST | &H8026110 |
| | 2 |
| PR_EMS_AB_AUTHORIZED_DOMAIN | &H8054001 |
| | E |
| PR_EMS_AB_AUTHORIZED_PASSWORD | &H8055010 |
| | 2 |

| | |
|--|----------------|
| PR_EMS_AB_AUTHORIZED_PASSWORD_CONFIRM | &H8192010
2 |
| PR_EMS_AB_AUTHORIZED_USER | &H8056001
E |
| PR_EMS_AB_AUTOREPLY | &H800B000
B |
| PR_EMS_AB_AUTOREPLY_MESSAGE | &H800A001
E |
| PR_EMS_AB_AUTOREPLY_SUBJECT | &H803E001
E |
| PR_EMS_AB_AVAILABLE_AUTHORIZATION_PACKAGES | &H8181101
E |
| PR_EMS_AB_AVAILABLE_DISTRIBUTIONS | &H818B001
E |
| PR_EMS_AB_BRIDGEHEAD_SERVERS | &H8174101
E |
| PR_EMS_AB_BRIDGEHEAD_SERVERS_O | &H8174000
D |
| PR_EMS_AB_BUSINESS_CATEGORY | &H8057101
E |
| PR_EMS_AB_BUSINESS_ROLES | &H8023010
2 |
| PR_EMS_AB_CA_CERTIFICATE | &H8003110
2 |
| PR_EMS_AB_CAN_CREATE_PF | &H8058101
E |
| PR_EMS_AB_CAN_CREATE_PF_BL | &H8059101
E |
| PR_EMS_AB_CAN_CREATE_PF_BL_O | &H8059000
D |
| PR_EMS_AB_CAN_CREATE_PF_DL | &H805A101
E |
| PR_EMS_AB_CAN_CREATE_PF_DL_BL | &H805B101
E |
| PR_EMS_AB_CAN_CREATE_PF_DL_BL_O | &H805B000
D |
| PR_EMS_AB_CAN_CREATE_PF_DL_O | &H805A000
D |
| PR_EMS_AB_CAN_CREATE_PF_O | &H8058000
D |
| PR_EMS_AB_CAN_NOT_CREATE_PF | &H805C101
E |
| PR_EMS_AB_CAN_NOT_CREATE_PF_BL | &H805D101
E |
| PR_EMS_AB_CAN_NOT_CREATE_PF_BL_O | &H805D000
D |
| PR_EMS_AB_CAN_NOT_CREATE_PF_DL | &H805E101
E |
| PR_EMS_AB_CAN_NOT_CREATE_PF_DL_BL | &H805F101 |

| | |
|--|---------------------|
| PR_EMS_AB_CAN_NOT_CREATE_PF_DL_BL_O | E
&H805F000
D |
| PR_EMS_AB_CAN_NOT_CREATE_PF_DL_O | &H805E000
D |
| PR_EMS_AB_CAN_NOT_CREATE_PF_O | &H805C000
D |
| PR_EMS_AB_CAN_PRESERVE_DNS | &H8060000
B |
| PR_EMS_AB_CERTIFICATE_CHAIN_V3 | &H8C27010
2 |
| PR_EMS_AB_CERTIFICATE_REVOCATION_LIST | &H8016010
2 |
| PR_EMS_AB_CERTIFICATE_REVOCATION_LIST_V1 | &H8C29010
2 |
| PR_EMS_AB_CERTIFICATE_REVOCATION_LIST_V3 | &H8C28010
2 |
| PR_EMS_AB_CHARACTER_SET | &H8185001
E |
| PR_EMS_AB_CHARACTER_SET_LIST | &H8182101
E |
| PR_EMS_AB_CHILD_RDNS | &HFFF8101
E |
| PR_EMS_AB_CLIENT_ACCESS_ENABLED | &H8C24000
B |
| PR_EMS_AB_CLOCK_ALERT_OFFSET | &H8061000
3 |
| PR_EMS_AB_CLOCK_ALERT_REPAIR | &H8062000
B |
| PR_EMS_AB_CLOCK_WARNING_OFFSET | &H8063000
3 |
| PR_EMS_AB_CLOCK_WARNING_REPAIR | &H8064000
B |
| PR_EMS_AB_COMPROMISED_KEY_LIST | &H81C4010
2 |
| PR_EMS_AB_COMPUTER_NAME | &H8065001
E |
| PR_EMS_AB_CONNECTED_DOMAINS | &H8066101
E |
| PR_EMS_AB_CONNECTION_LIST_FILTER | &H8180010
2 |
| PR_EMS_AB_CONNECTION_LIST_FILTER_TYPE | &H81B4000
3 |
| PR_EMS_AB_CONNECTION_TYPE | &H81B3000
B |
| PR_EMS_AB_CONTAINER_INFO | &H8067000
3 |
| PR_EMS_AB_CONTAINERID | &HFFFD000
3 |

| | |
|------------------------------------|----------------|
| PR_EMS_AB_CONTENT_TYPE | &H8186000
3 |
| PR_EMS_AB_CONTROL_MSG_FOLDER_ID | &H8188010
2 |
| PR_EMS_AB_CONTROL_MSG_RULES | &H818A010
2 |
| PR_EMS_AB_COST | &H8068000
3 |
| PR_EMS_AB_COUNTRY_NAME | &H8069001
E |
| PR_EMS_AB_CROSS_CERTIFICATE_CRL | &H8C30110
2 |
| PR_EMS_AB_CROSS_CERTIFICATE_PAIR | &H8025110
2 |
| PR_EMS_AB_DEFAULT_MESSAGE_FORMAT | &H8C37000
B |
| PR_EMS_AB_DELEGATE_USER | &H8C49000
B |
| PR_EMS_AB_DELIV_CONT_LENGTH | &H806A000
3 |
| PR_EMS_AB_DELIV_EITS | &H806B110
2 |
| PR_EMS_AB_DELIV_EXT_CONT_TYPES | &H806C110
2 |
| PR_EMS_AB_DELIVER_AND_REDIRECT | &H806D000
B |
| PR_EMS_AB_DELIVERY_MECHANISM | &H806E000
3 |
| PR_EMS_AB_DELTA_REVOCATION_LIST | &H8C46110
2 |
| PR_EMS_AB_DESCRIPTION | &H806F101
E |
| PR_EMS_AB_DESTINATION_INDICATOR | &H8070101
E |
| PR_EMS_AB_DIAGNOSTIC_REG_KEY | &H8071001
E |
| PR_EMS_AB_DISABLE_DEFERRED_COMMIT | &H8C23000
B |
| PR_EMS_AB_DISABLED_GATEWAY_PROXY | &H81C3101
E |
| PR_EMS_AB_DISPLAY_NAME_OVERRIDE | &H8001000
B |
| PR_EMS_AB_DISPLAY_NAME_PRINTABLE | &H39FF001
E |
| PR_EMS_AB_DISPLAY_NAME_SUFFIX | &H8C44001
E |
| PR_EMS_AB_DL_MEM_REJECT_PERMS_BL | &H8072101
E |
| PR_EMS_AB_DL_MEM_REJECT_PERMS_BL_O | &H8072000 |

| | |
|-------------------------------------|-----------|
| | D |
| PR_EMS_AB_DL_MEM_SUBMIT_PERMS_BL | &H8073101 |
| | E |
| PR_EMS_AB_DL_MEM_SUBMIT_PERMS_BL_O | &H8073000 |
| | D |
| PR_EMS_AB_DL_MEMBER_RULE | &H8074110 |
| | 2 |
| PR_EMS_AB_DMD_NAME | &H8C56001 |
| | E |
| PR_EMS_AB_DO_OAB_VERSION | &H8C3A000 |
| | 3 |
| PR_EMS_AB_DOMAIN_DEF_ALT_RECIP | &H8075001 |
| | E |
| PR_EMS_AB_DOMAIN_DEF_ALT_RECIP_O | &H8075000 |
| | D |
| PR_EMS_AB_DOMAIN_NAME | &H8076001 |
| | E |
| PR_EMS_AB_DOS_ENTRYID | &HFFFD000 |
| | 3 |
| PR_EMS_AB_DSA_SIGNATURE | &H8077010 |
| | 2 |
| PR_EMS_AB_DXA_ADMIN_COPY | &H8078000 |
| | B |
| PR_EMS_AB_DXA_ADMIN_FORWARD | &H8079000 |
| | B |
| PR_EMS_AB_DXA_ADMIN_UPDATE | &H807A000 |
| | 3 |
| PR_EMS_AB_DXA_APPEND_REQCN | &H807B000 |
| | B |
| PR_EMS_AB_DXA_CONF_CONTAINER_LIST | &H807C101 |
| | E |
| PR_EMS_AB_DXA_CONF_CONTAINER_LIST_O | &H807C000 |
| | D |
| PR_EMS_AB_DXA_CONF_REQ_TIME | &H807D004 |
| | 0 |
| PR_EMS_AB_DXA_CONF_SEQ | &H807E001 |
| | E |
| PR_EMS_AB_DXA_CONF_SEQ_USN | &H807F000 |
| | 3 |
| PR_EMS_AB_DXA_EXCHANGE_OPTIONS | &H8080000 |
| | 3 |
| PR_EMS_AB_DXA_EXPORT_NOW | &H8081000 |
| | B |
| PR_EMS_AB_DXA_FLAGS | &H8082000 |
| | 3 |
| PR_EMS_AB_DXA_IMP_SEQ | &H8083001 |
| | E |
| PR_EMS_AB_DXA_IMP_SEQ_TIME | &H8084004 |
| | 0 |

| | |
|--|----------------|
| PR_EMS_AB_DXA_IMP_SEQ_USN | &H8085000
3 |
| PR_EMS_AB_DXA_IMPORT_NOW | &H8086000
B |
| PR_EMS_AB_DXA_IN_TEMPLATE_MAP | &H8087101
E |
| PR_EMS_AB_DXA_LOCAL_ADMIN | &H8088001
E |
| PR_EMS_AB_DXA_LOCAL_ADMIN_O | &H8088000
D |
| PR_EMS_AB_DXA_LOGGING_LEVEL | &H8089000
3 |
| PR_EMS_AB_DXA_NATIVE_ADDRESS_TYPE | &H808A001
E |
| PR_EMS_AB_DXA_OUT_TEMPLATE_MAP | &H808B101
E |
| PR_EMS_AB_DXA_PASSWORD | &H808C001
E |
| PR_EMS_AB_DXA_PREV_EXCHANGE_OPTIONS | &H808D000
3 |
| PR_EMS_AB_DXA_PREV_EXPORT_NATIVE_ONLY | &H808E000
B |
| PR_EMS_AB_DXA_PREV_IN_EXCHANGE_SENSITIVITY | &H808F000
3 |
| PR_EMS_AB_DXA_PREV_REMOTE_ENTRIES | &H8090001
E |
| PR_EMS_AB_DXA_PREV_REMOTE_ENTRIES_O | &H8090000
D |
| PR_EMS_AB_DXA_PREV_REPLICATION_SENSITIVITY | &H8091000
3 |
| PR_EMS_AB_DXA_PREV_TEMPLATE_OPTIONS | &H8092000
3 |
| PR_EMS_AB_DXA_PREV_TYPES | &H8093000
3 |
| PR_EMS_AB_DXA_RECIPIENT_CP | &H8094001
E |
| PR_EMS_AB_DXA_REMOTE_CLIENT | &H8095001
E |
| PR_EMS_AB_DXA_REMOTE_CLIENT_O | &H8095000
D |
| PR_EMS_AB_DXA_REQ_SEQ | &H8096001
E |
| PR_EMS_AB_DXA_REQ_SEQ_TIME | &H8097004
0 |
| PR_EMS_AB_DXA_REQ_SEQ_USN | &H8098000
3 |
| PR_EMS_AB_DXA_REQNAME | &H8099001
E |
| PR_EMS_AB_DXA_SVR_SEQ | &H809A001 |

| | |
|--|---------------------|
| PR_EMS_AB_DXA_SVR_SEQ_TIME | E
&H809B004
0 |
| PR_EMS_AB_DXA_SVR_SEQ_USN | &H809C000
3 |
| PR_EMS_AB_DXA_TASK | &H809D000
3 |
| PR_EMS_AB_DXA_TEMPLATE_OPTIONS | &H809E000
3 |
| PR_EMS_AB_DXA_TEMPLATE_TIMESTAMP | &H809F004
0 |
| PR_EMS_AB_DXA_TYPES | &H80A0000
3 |
| PR_EMS_AB_DXA_UNCONF_CONTAINER_LIST | &H80A1101
E |
| PR_EMS_AB_DXA_UNCONF_CONTAINER_LIST_O | &H80A1000
D |
| PR_EMS_AB_EMPLOYEE_NUMBER | &H8C67001
E |
| PR_EMS_AB_EMPLOYEE_TYPE | &H8C69001
E |
| PR_EMS_AB_ENABLE_COMPATIBILITY | &H8C32000
B |
| PR_EMS_AB_ENABLED | &H8C21000
B |
| PR_EMS_AB_ENABLED_AUTHORIZATION_PACKAGES | &H8184101
E |
| PR_EMS_AB_ENABLED_PROTOCOL_CFG | &H81A8000
B |
| PR_EMS_AB_ENABLED_PROTOCOLS | &H817F000
3 |
| PR_EMS_AB_ENCAPSULATION_METHOD | &H80A2000
3 |
| PR_EMS_AB_ENCRYPT | &H80A3000
B |
| PR_EMS_AB_ENCRYPT_ALG_LIST_NA | &H8040101
E |
| PR_EMS_AB_ENCRYPT_ALG_LIST_OTHER | &H8041101
E |
| PR_EMS_AB_ENCRYPT_ALG_SELECTED_NA | &H8043001
E |
| PR_EMS_AB_ENCRYPT_ALG_SELECTED_OTHER | &H803D001
E |
| PR_EMS_AB_EXPAND_DLS_LOCALLY | &H80A4000
B |
| PR_EMS_AB_EXPIRATION_TIME | &H8028004
0 |
| PR_EMS_AB_EXPORT_CONTAINERS | &H80A5101
E |

| | |
|--------------------------------------|----------------|
| PR_EMS_AB_EXPORT_CONTAINERS_O | &H80A5000
D |
| PR_EMS_AB_EXPORT_CUSTOM_RECIPIENTS | &H80A6000
B |
| PR_EMS_AB_EXTENDED_CHARS_ALLOWED | &H80A7000
B |
| PR_EMS_AB_EXTENSION_ATTRIBUTE_1 | &H802D001
E |
| PR_EMS_AB_EXTENSION_ATTRIBUTE_10 | &H8036001
E |
| PR_EMS_AB_EXTENSION_ATTRIBUTE_11 | &H8C57001
E |
| PR_EMS_AB_EXTENSION_ATTRIBUTE_12 | &H8C58001
E |
| PR_EMS_AB_EXTENSION_ATTRIBUTE_13 | &H8C59001
E |
| PR_EMS_AB_EXTENSION_ATTRIBUTE_14 | &H8C60001
E |
| PR_EMS_AB_EXTENSION_ATTRIBUTE_15 | &H8C61001
E |
| PR_EMS_AB_EXTENSION_ATTRIBUTE_2 | &H802E001
E |
| PR_EMS_AB_EXTENSION_ATTRIBUTE_3 | &H802F001
E |
| PR_EMS_AB_EXTENSION_ATTRIBUTE_4 | &H8030001
E |
| PR_EMS_AB_EXTENSION_ATTRIBUTE_5 | &H8031001
E |
| PR_EMS_AB_EXTENSION_ATTRIBUTE_6 | &H8032001
E |
| PR_EMS_AB_EXTENSION_ATTRIBUTE_7 | &H8033001
E |
| PR_EMS_AB_EXTENSION_ATTRIBUTE_8 | &H8034001
E |
| PR_EMS_AB_EXTENSION_ATTRIBUTE_9 | &H8035001
E |
| PR_EMS_AB_EXTENSION_DATA | &H80A8110
2 |
| PR_EMS_AB_EXTENSION_NAME | &H80A9101
E |
| PR_EMS_AB_EXTENSION_NAME_INHERITED | &H80AA101
E |
| PR_EMS_AB_FACSIMILE_TELEPHONE_NUMBER | &H80AB110
2 |
| PR_EMS_AB_FILE_VERSION | &H80AC010
2 |
| PR_EMS_AB_FILTER_LOCAL_ADDRESSES | &H80AD000
B |
| PR_EMS_AB_FOLDER_PATHNAME | &H8004001 |

| | |
|------------------------------------|-----------|
| | E |
| PR_EMS_AB_FOLDERS_CONTAINER | &H80AE001 |
| | E |
| PR_EMS_AB_FOLDERS_CONTAINER_O | &H80AE000 |
| | D |
| PR_EMS_AB_FORM_DATA | &H8C65010 |
| | 2 |
| PR_EMS_AB_FORWARDING_ADDRESS | &H8C64001 |
| | E |
| PR_EMS_AB_GARBAGE_COLL_PERIOD | &H80AF000 |
| | 3 |
| PR_EMS_AB_GATEWAY_LOCAL_CRED | &H80B0001 |
| | E |
| PR_EMS_AB_GATEWAY_LOCAL_DESIG | &H80B1001 |
| | E |
| PR_EMS_AB_GATEWAY_PROXY | &H80B2101 |
| | E |
| PR_EMS_AB_GATEWAY_ROUTING_TREE | &H80B3010 |
| | 2 |
| PR_EMS_AB_GENERATION_QUALIFIER | &H8C53001 |
| | E |
| PR_EMS_AB_GROUP_BY_ATTR_1 | &H81B7001 |
| | E |
| PR_EMS_AB_GROUP_BY_ATTR_2 | &H81B8001 |
| | E |
| PR_EMS_AB_GROUP_BY_ATTR_3 | &H81B9001 |
| | E |
| PR_EMS_AB_GROUP_BY_ATTR_4 | &H81BA001 |
| | E |
| PR_EMS_AB_GROUP_BY_ATTR_VALUE_DN | &H8C1A001 |
| | E |
| PR_EMS_AB_GROUP_BY_ATTR_VALUE_DN_O | &H8C1A000 |
| | D |
| PR_EMS_AB_GROUP_BY_ATTR_VALUE_STR | &H8C19001 |
| | E |
| PR_EMS_AB_GWART_LAST_MODIFIED | &H80B4004 |
| | 0 |
| PR_EMS_AB_HAS_FULL_REPLICA_NCS | &H80B5101 |
| | E |
| PR_EMS_AB_HAS_FULL_REPLICA_NCS_O | &H80B5000 |
| | D |
| PR_EMS_AB_HAS_MASTER_NCS | &H80B6101 |
| | E |
| PR_EMS_AB_HAS_MASTER_NCS_O | &H80B6000 |
| | D |
| PR_EMS_AB_HELP_DATA16 | &H803A010 |
| | 2 |
| PR_EMS_AB_HELP_DATA32 | &H8010010 |
| | 2 |

| | |
|----------------------------------|----------------|
| PR_EMS_AB_HELP_FILE_NAME | &H803B001
E |
| PR_EMS_AB_HEURISTICS | &H80B7000
3 |
| PR_EMS_AB_HIDE_DL_MEMBERSHIP | &H80B8000
B |
| PR_EMS_AB_HIDE_FROM_ADDRESS_BOOK | &H80B9000
B |
| PR_EMS_AB_HIERARCHY_PATH | &HFFF9001
E |
| PR_EMS_AB_HOME_MDB | &H8006001
E |
| PR_EMS_AB_HOME_MDB_BL | &H8014101
E |
| PR_EMS_AB_HOME_MDB_BL_O | &H8014000
D |
| PR_EMS_AB_HOME_MDB_O | &H8006000
D |
| PR_EMS_AB_HOME_MTA | &H8007001
E |
| PR_EMS_AB_HOME_MTA_O | &H8007000
D |
| PR_EMS_AB_HOME_PUBLIC_SERVER | &H803F001
E |
| PR_EMS_AB_HOME_PUBLIC_SERVER_O | &H803F000
D |
| PR_EMS_AB_HOUSE_IDENTIFIER | &H8C54001
E |
| PR_EMS_AB_HTTP_PUB_AB_ATTRIBUTES | &H81A9101
E |
| PR_EMS_AB_HTTP_PUB_GAL | &H819B000
B |
| PR_EMS_AB_HTTP_PUB_GAL_LIMIT | &H819C000
3 |
| PR_EMS_AB_HTTP_PUB_PF | &H819E110
2 |
| PR_EMS_AB_HTTP_SERVERS | &H81AB101
E |
| PR_EMS_AB_IMPORT_CONTAINER | &H80BA001
E |
| PR_EMS_AB_IMPORT_CONTAINER_O | &H80BA000
D |
| PR_EMS_AB_IMPORT_SENSITIVITY | &H80BB000
3 |
| PR_EMS_AB_IMPORTED_FROM | &H8042001
E |
| PR_EMS_AB_INBOUND_ACCEPT_ALL | &H8C20000
B |
| PR_EMS_AB_INBOUND_DN | &H8C1E001 |

| | |
|-------------------------------------|----------------|
| | E |
| PR_EMS_AB_INBOUND_DN_O | &H8C1E000
D |
| PR_EMS_AB_INBOUND_HOST | &H818E101
E |
| PR_EMS_AB_INBOUND_NEWSFEED | &H8193001
E |
| PR_EMS_AB_INBOUND_NEWSFEED_TYPE | &H8C1F000
B |
| PR_EMS_AB_INBOUND_SITES | &H80BC101
E |
| PR_EMS_AB_INBOUND_SITES_O | &H80BC000
D |
| PR_EMS_AB_INCOMING_MSG_SIZE_LIMIT | &H8190000
3 |
| PR_EMS_AB_INCOMING_PASSWORD | &H81AF010
2 |
| PR_EMS_AB_INSADMIN | &H81C5001
E |
| PR_EMS_AB_INSADMIN_O | &H81C5000
D |
| PR_EMS_AB_INSTANCE_TYPE | &H80BD000
3 |
| PR_EMS_AB_INTERNATIONAL_ISDN_NUMBER | &H80BE101
E |
| PR_EMS_AB_INVOCATION_ID | &H80BF010
2 |
| PR_EMS_AB_IS_DELETED | &H80C0000
B |
| PR_EMS_AB_IS_MASTER | &HFFF000
B |
| PR_EMS_AB_IS_MEMBER_OF_DL | &H8008000
D |
| PR_EMS_AB_IS_MEMBER_OF_DL_T | &H8008001
E |
| PR_EMS_AB_IS_SINGLE_VALUED | &H80C1000
B |
| PR_EMS_AB_KCC_STATUS | &H80C2110
2 |
| PR_EMS_AB_KM_SERVER | &H800D001
E |
| PR_EMS_AB_KM_SERVER_O | &H800D000
D |
| PR_EMS_AB_KNOWLEDGE_INFORMATION | &H80C3101
E |
| PR_EMS_AB_LABELEDURI | &H8C51001
E |
| PR_EMS_AB_LANGUAGE | &H8178000
3 |

| | |
|--------------------------------------|----------------|
| PR_EMS_AB_LANGUAGE_ISO639 | &H8C6C001
E |
| PR_EMS_AB_LDAP_DISPLAY_NAME | &H8171101
E |
| PR_EMS_AB_LDAP_SEARCH_CFG | &H8C1D000
3 |
| PR_EMS_AB_LINE_WRAP | &H80C4000
3 |
| PR_EMS_AB_LINK_ID | &H80C5000
3 |
| PR_EMS_AB_LIST_PUBLIC_FOLDERS | &H8C50000
B |
| PR_EMS_AB_LOCAL_BRIDGE_HEAD | &H80C6001
E |
| PR_EMS_AB_LOCAL_BRIDGE_HEAD_ADDRESS | &H80C7001
E |
| PR_EMS_AB_LOCAL_INITIAL_TURN | &H80C8000
B |
| PR_EMS_AB_LOCAL_SCOPE | &H80C9101
E |
| PR_EMS_AB_LOCAL_SCOPE_O | &H80C9000
D |
| PR_EMS_AB_LOG_FILENAME | &H80CA001
E |
| PR_EMS_AB_LOG_ROLLOVER_INTERVAL | &H80CB000
3 |
| PR_EMS_AB_MAIL_DROP | &H8C63001
E |
| PR_EMS_AB_MAINTAIN_AUTOREPLY_HISTORY | &H80CC000
B |
| PR_EMS_AB_MANAGER | &H8005000
D |
| PR_EMS_AB_MANAGER_T | &H8005001
E |
| PR_EMS_AB_MAPI_DISPLAY_TYPE | &H80CD000
3 |
| PR_EMS_AB_MAPI_ID | &H80CE000
3 |
| PR_EMS_AB_MAXIMUM_OBJECT_ID | &H8169010
2 |
| PR_EMS_AB_MDB_BACKOFF_INTERVAL | &H80CF000
3 |
| PR_EMS_AB_MDB_MSG_TIME_OUT_PERIOD | &H80D0000
3 |
| PR_EMS_AB_MDB_OVER_QUOTA_LIMIT | &H80D1000
3 |
| PR_EMS_AB_MDB_STORAGE_QUOTA | &H80D2000
3 |
| PR_EMS_AB_MDB_UNREAD_LIMIT | &H80D3000 |

| | |
|---|----------------|
| | 3 |
| PR_EMS_AB_MDB_USE_DEFAULTS | &H80D4000
B |
| PR_EMS_AB_MEMBER | &H8009000
D |
| PR_EMS_AB_MEMBER_T | &H8009001
E |
| PR_EMS_AB_MESSAGE_TRACKING_ENABLED | &H80D5000
B |
| PR_EMS_AB_MIME_TYPES | &H8C1C010
2 |
| PR_EMS_AB_MODERATED | &H81AC000
B |
| PR_EMS_AB_MODERATOR | &H8199001
E |
| PR_EMS_AB_MONITOR_CLOCK | &H80D6000
B |
| PR_EMS_AB_MONITOR_SERVERS | &H80D7000
B |
| PR_EMS_AB_MONITOR_SERVICES | &H80D8000
B |
| PR_EMS_AB_MONITORED_CONFIGURATIONS | &H80D9101
E |
| PR_EMS_AB_MONITORED_CONFIGURATIONS_O | &H80D9000
D |
| PR_EMS_AB_MONITORED_SERVERS | &H80DA101
E |
| PR_EMS_AB_MONITORED_SERVERS_O | &H80DA000
D |
| PR_EMS_AB_MONITORED_SERVICES | &H80DB101
E |
| PR_EMS_AB_MONITORING_ALERT_DELAY | &H80DC000
3 |
| PR_EMS_AB_MONITORING_ALERT_UNITS | &H80DD000
3 |
| PR_EMS_AB_MONITORING_AVAILABILITY_STYLE | &H80DE000
3 |
| PR_EMS_AB_MONITORING_AVAILABILITY_WINDOW | &H80DF010
2 |
| PR_EMS_AB_MONITORING_CACHED_VIA_MAIL | &H80E0101
E |
| PR_EMS_AB_MONITORING_CACHED_VIA_MAIL_O | &H80E0000
D |
| PR_EMS_AB_MONITORING_CACHED_VIA_RPC | &H80E1101
E |
| PR_EMS_AB_MONITORING_CACHED_VIA_RPC_O | &H80E1000
D |
| PR_EMS_AB_MONITORING_ESCALATION_PROCEDURE | &H80E2110
2 |

| | |
|--|----------------|
| PR_EMS_AB_MONITORING_HOTSITE_POLL_INTERVAL | &H80E3000
3 |
| PR_EMS_AB_MONITORING_HOTSITE_POLL_UNITS | &H80E4000
3 |
| PR_EMS_AB_MONITORING_MAIL_UPDATE_INTERVAL | &H80E5000
3 |
| PR_EMS_AB_MONITORING_MAIL_UPDATE_UNITS | &H80E6000
3 |
| PR_EMS_AB_MONITORING_NORMAL_POLL_INTERVAL | &H80E7000
3 |
| PR_EMS_AB_MONITORING_NORMAL_POLL_UNITS | &H80E8000
3 |
| PR_EMS_AB_MONITORING_RECIPIENTS | &H80E9101
E |
| PR_EMS_AB_MONITORING_RECIPIENTS_NDR | &H80EA101
E |
| PR_EMS_AB_MONITORING_RECIPIENTS_NDR_O | &H80EA000
D |
| PR_EMS_AB_MONITORING_RECIPIENTS_O | &H80E9000
D |
| PR_EMS_AB_MONITORING_RPC_UPDATE_INTERVAL | &H80EB000
3 |
| PR_EMS_AB_MONITORING_RPC_UPDATE_UNITS | &H80EC000
3 |
| PR_EMS_AB_MONITORING_WARNING_DELAY | &H80ED000
3 |
| PR_EMS_AB_MONITORING_WARNING_UNITS | &H80EE000
3 |
| PR_EMS_AB_MTA_LOCAL_CRED | &H80EF001
E |
| PR_EMS_AB_MTA_LOCAL_DESIG | &H80F0001
E |
| PR_EMS_AB_N_ADDRESS | &H80F1010
2 |
| PR_EMS_AB_N_ADDRESS_TYPE | &H80F2000
3 |
| PR_EMS_AB_NETWORK_ADDRESS | &H8170101
E |
| PR_EMS_AB_NEWSFEED_TYPE | &H8194000
3 |
| PR_EMS_AB_NEWSGROUP | &H8198001
E |
| PR_EMS_AB_NEWSGROUP_LIST | &H8196010
2 |
| PR_EMS_AB_NNTP_CHARACTER_SET | &H817D001
E |
| PR_EMS_AB_NNTP_CONTENT_FORMAT | &H8176001
E |
| PR_EMS_AB_NNTP_DISTRIBUTIONS | &H8197101 |

| | |
|------------------------------------|---------------------|
| PR_EMS_AB_NNTP_DISTRIBUTIONS_FLAG | E
&H81A4000
B |
| PR_EMS_AB_NNTP_NEWSFEEDS | &H81A6101
E |
| PR_EMS_AB_NNTP_NEWSFEEDS_O | &H81A6000
D |
| PR_EMS_AB_NT_MACHINE_NAME | &H80F3001
E |
| PR_EMS_AB_NT_SECURITY_DESCRIPTOR | &H8013010
2 |
| PR_EMS_AB_NUM_OF_OPEN_RETRIES | &H80F4000
3 |
| PR_EMS_AB_NUM_OF_TRANSFER_RETRIES | &H80F5000
3 |
| PR_EMS_AB_OBJ_DIST_NAME | &H803C001
E |
| PR_EMS_AB_OBJ_DIST_NAME_O | &H803C000
D |
| PR_EMS_AB_OBJ_VIEW_CONTAINERS | &H81C7101
E |
| PR_EMS_AB_OBJ_VIEW_CONTAINERS_O | &H81C7000
D |
| PR_EMS_AB_OBJECT_CLASS_CATEGORY | &H80F6000
3 |
| PR_EMS_AB_OBJECT_OID | &HFFFA010
2 |
| PR_EMS_AB_OBJECT_VERSION | &H80F7000
3 |
| PR_EMS_AB_OFF_LINE_AB_CONTAINERS | &H80F8101
E |
| PR_EMS_AB_OFF_LINE_AB_CONTAINERS_O | &H80F8000
D |
| PR_EMS_AB_OFF_LINE_AB_SCHEDULE | &H80F9010
2 |
| PR_EMS_AB_OFF_LINE_AB_SERVER | &H80FA001
E |
| PR_EMS_AB_OFF_LINE_AB_SERVER_O | &H80FA000
D |
| PR_EMS_AB_OFF_LINE_AB_STYLE | &H80FB000
3 |
| PR_EMS_AB_OID_TYPE | &H80FC000
3 |
| PR_EMS_AB_OM_OBJECT_CLASS | &H80FD010
2 |
| PR_EMS_AB_OM_SYNTAX | &H80FE000
3 |
| PR_EMS_AB_OOF_REPLY_TO_ORIGINATOR | &H80FF000
B |

| | |
|--|----------------|
| PR_EMS_AB_OPEN_RETRY_INTERVAL | &H8100000
3 |
| PR_EMS_AB_ORGANIZATION_NAME | &H8101101
E |
| PR_EMS_AB_ORGANIZATIONAL_UNIT_NAME | &H8102101
E |
| PR_EMS_AB_ORIGINAL_DISPLAY_TABLE | &H8103010
2 |
| PR_EMS_AB_ORIGINAL_DISPLAY_TABLE_MSDOS | &H8104010
2 |
| PR_EMS_AB_OTHER_RECIPS | &HF000000
D |
| PR_EMS_AB_OUTBOUND_HOST | &H818D010
2 |
| PR_EMS_AB_OUTBOUND_HOST_TYPE | &H81B0000
B |
| PR_EMS_AB_OUTBOUND_NEWSFEED | &H8195001
E |
| PR_EMS_AB_OUTBOUND_SITES | &H8105101
E |
| PR_EMS_AB_OUTBOUND_SITES_O | &H8105000
D |
| PR_EMS_AB_OUTGOING_MSG_SIZE_LIMIT | &H818F000
3 |
| PR_EMS_AB_OVERRIDE_NNTP_CONTENT_FORMAT | &H81C6000
B |
| PR_EMS_AB_OWA_SERVER | &H8C66001
E |
| PR_EMS_AB_OWNER | &H800C001
E |
| PR_EMS_AB_OWNER_BL | &H8024101
E |
| PR_EMS_AB_OWNER_BL_O | &H8024000
D |
| PR_EMS_AB_OWNER_O | &H800C000
D |
| PR_EMS_AB_P_SELECTOR | &H8106010
2 |
| PR_EMS_AB_P_SELECTOR_INBOUND | &H8107010
2 |
| PR_EMS_AB_PARENT_ENTRYID | &HFFFC010
2 |
| PR_EMS_AB_PER_MSG_DIALOG_DISPLAY_TABLE | &H8108010
2 |
| PR_EMS_AB_PER_RECIP_DIALOG_DISPLAY_TABLE | &H8109010
2 |
| PR_EMS_AB_PERIOD_REP_SYNC_TIMES | &H810A010
2 |
| PR_EMS_AB_PERIOD_REPL_STAGGER | &H810B000 |

| | |
|---------------------------------------|----------------|
| | 3 |
| PR_EMS_AB_PERSONAL_TITLE | &H8C6B001
E |
| PR_EMS_AB_PF_CONTACTS | &H8038101
E |
| PR_EMS_AB_PF_CONTACTS_O | &H8038000
D |
| PR_EMS_AB_POP_CHARACTER_SET | &H8179001
E |
| PR_EMS_AB_POP_CONTENT_FORMAT | &H8177001
E |
| PR_EMS_AB_PORT_NUMBER | &H81B5000
3 |
| PR_EMS_AB_POSTAL_ADDRESS | &H810C110
2 |
| PR_EMS_AB_PREFERRED_DELIVERY_METHOD | &H810D100
3 |
| PR_EMS_AB_PRESERVE_INTERNET_CONTENT | &H8C22000
B |
| PR_EMS_AB_PRMD | &H810E001
E |
| PR_EMS_AB_PROMO_EXPIRATION | &H81C2004
0 |
| PR_EMS_AB_PROTOCOL_SETTINGS | &H81B6101
E |
| PR_EMS_AB_PROXY_ADDRESSES | &H800F101
E |
| PR_EMS_AB_PROXY_GENERATION_ENABLED | &H81B1000
B |
| PR_EMS_AB_PROXY_GENERATOR_DLL | &H810F001
E |
| PR_EMS_AB_PUBLIC_DELEGATES | &H8015000
D |
| PR_EMS_AB_PUBLIC_DELEGATES_BL | &H8110101
E |
| PR_EMS_AB_PUBLIC_DELEGATES_BL_O | &H8110000
D |
| PR_EMS_AB_PUBLIC_DELEGATES_T | &H8015001
E |
| PR_EMS_AB_QUOTA_NOTIFICATION_SCHEDULE | &H81110102 |
| PR_EMS_AB_QUOTA_NOTIFICATION_STYLE | &H8112000
3 |
| PR_EMS_AB_RANGE_LOWER | &H8113000
3 |
| PR_EMS_AB_RANGE_UPPER | &H8114000
3 |
| PR_EMS_AB_RAS_ACCOUNT | &H81AD001
E |
| PR_EMS_AB_RAS_CALLBACK_NUMBER | &H8115001 |

| | |
|--------------------------------------|----------------|
| | E |
| PR_EMS_AB_RAS_PASSWORD | &H81AE010
2 |
| PR_EMS_AB_RAS_PHONE_NUMBER | &H8116001
E |
| PR_EMS_AB_RAS_PHONEBOOK_ENTRY_NAME | &H8117001
E |
| PR_EMS_AB_RAS_REMOTE_SRVR_NAME | &H8118001
E |
| PR_EMS_AB_REFERRAL_LIST | &H81A3101
E |
| PR_EMS_AB_REGISTERED_ADDRESS | &H81191102 |
| PR_EMS_AB_REMOTE_BRIDGE_HEAD | &H811A001
E |
| PR_EMS_AB_REMOTE_BRIDGE_HEAD_ADDRESS | &H811B001
E |
| PR_EMS_AB_REMOTE_OUT_BH_SERVER | &H811C001
E |
| PR_EMS_AB_REMOTE_OUT_BH_SERVER_O | &H811C000
D |
| PR_EMS_AB_REMOTE_SITE | &H811D001
E |
| PR_EMS_AB_REMOTE_SITE_O | &H811D000
D |
| PR_EMS_AB_REPLICATED_OBJECT_VERSION | &H8C62000
3 |
| PR_EMS_AB_REPLICATION_MAIL_MSG_SIZE | &H8168000
3 |
| PR_EMS_AB_REPLICATION_SENSITIVITY | &H811E000
3 |
| PR_EMS_AB_REPLICATION_STAGGER | &H811F000
3 |
| PR_EMS_AB_REPORT_TO_ORIGINATOR | &H8120000
B |
| PR_EMS_AB_REPORT_TO_OWNER | &H8121000
B |
| PR_EMS_AB_REPORTS | &H800E000
D |
| PR_EMS_AB_REPORTS_T | &H800E000
D |
| PR_EMS_AB_REQ_SEQ | &H8122000
3 |
| PR_EMS_AB_REQUIRE_SSL | &H8C25000
B |
| PR_EMS_AB_RESPONSIBLE_LOCAL_DXA | &H8123001
E |
| PR_EMS_AB_RESPONSIBLE_LOCAL_DXA_O | &H8123000
D |
| PR_EMS_AB_RETURN_EXACT_MSG_SIZE | &H8C52000 |

| | |
|-------------------------------------|---------------------|
| PR_EMS_AB_RID_SERVER | B
&H8124001
E |
| PR_EMS_AB_RID_SERVER_O | &H8124000
D |
| PR_EMS_AB_ROLE_OCCUPANT | &H8125101
E |
| PR_EMS_AB_ROLE_OCCUPANT_O | &H8125000
D |
| PR_EMS_AB_ROOT_NEWSGROUPS_FOLDER_ID | &H81B2010
2 |
| PR_EMS_AB_ROUTING_LIST | &H8126101
E |
| PR_EMS_AB_RTS_CHECKPOINT_SIZE | &H8127000
3 |
| PR_EMS_AB_RTS_RECOVERY_TIMEOUT | &H8128000
3 |
| PR_EMS_AB_RTS_WINDOW_SIZE | &H8129000
3 |
| PR_EMS_AB_RUNS_ON | &H812A101
E |
| PR_EMS_AB_RUNS_ON_O | &H812A000
D |
| PR_EMS_AB_S_SELECTOR | &H812B010
2 |
| PR_EMS_AB_S_SELECTOR_INBOUND | &H812C010
2 |
| PR_EMS_AB_SCHEMA_FLAGS | &H8173000
3 |
| PR_EMS_AB_SCHEMA_VERSION | &H817C100
3 |
| PR_EMS_AB_SEARCH_FLAGS | &H812D000
3 |
| PR_EMS_AB_SEARCH_GUIDE | &H812E110
2 |
| PR_EMS_AB_SECURITY_POLICY | &H8C47110
2 |
| PR_EMS_AB_SECURITY_PROTOCOL | &H8037110
2 |
| PR_EMS_AB_SEE_ALSO | &H812F101
E |
| PR_EMS_AB_SEE_ALSO_O | &H812F000
D |
| PR_EMS_AB_SEND_EMAIL_MESSAGE | &H8C31000
B |
| PR_EMS_AB_SEND_TNEF | &H8191000
B |
| PR_EMS_AB_SERIAL_NUMBER | &H8130101
E |

| | |
|---|----------------|
| PR_EMS_AB_SERVER | &HFFFE001
E |
| PR_EMS_AB_SERVICE_ACTION_FIRST | &H8131000
3 |
| PR_EMS_AB_SERVICE_ACTION_OTHER | &H8132000
3 |
| PR_EMS_AB_SERVICE_ACTION_SECOND | &H8133000
3 |
| PR_EMS_AB_SERVICE_RESTART_DELAY | &H8134000
3 |
| PR_EMS_AB_SERVICE_RESTART_MESSAGE | &H8135001
E |
| PR_EMS_AB_SESSION_DISCONNECT_TIMER | &H8136000
3 |
| PR_EMS_AB_SITE_AFFINITY | &H8137101
E |
| PR_EMS_AB_SITE_FOLDER_GUID | &H8166010
2 |
| PR_EMS_AB_SITE_FOLDER_SERVER | &H8167001
E |
| PR_EMS_AB_SITE_FOLDER_SERVER_O | &H8167000
D |
| PR_EMS_AB_SITE_PROXY_SPACE | &H8138101
E |
| PR_EMS_AB_SMIME_ALG_LIST_NA | &H8C33101
E |
| PR_EMS_AB_SMIME_ALG_LIST_OTHER | &H8C34101
E |
| PR_EMS_AB_SMIME_ALG_SELECTED_NA | &H8C35001
E |
| PR_EMS_AB_SMIME_ALG_SELECTED_OTHER | &H8C36001
E |
| PR_EMS_AB_SPACE_LAST_COMPUTED | &H8139004
0 |
| PR_EMS_AB_STREET_ADDRESS | &H813A001
E |
| PR_EMS_AB_SUB_REFS | &H813B101
E |
| PR_EMS_AB_SUB_REFS_O | &H813B000
D |
| PR_EMS_AB_SUB_SITE | &H817B001
E |
| PR_EMS_AB_SUBMISSION_CONT_LENGTH | &H813C000
3 |
| PR_EMS_AB_SUPPORT_SMIME_SIGNATURES | &H8C48000
B |
| PR_EMS_AB_SUPPORTED_ALGORITHMS | &H8C55010
2 |
| PR_EMS_AB_SUPPORTED_APPLICATION_CONTEXT | &H813D110 |

| | |
|---------------------------------------|----------------|
| | 2 |
| PR_EMS_AB_SUPPORTING_STACK | &H813E101
E |
| PR_EMS_AB_SUPPORTING_STACK_BL | &H813F101
E |
| PR_EMS_AB_SUPPORTING_STACK_BL_O | &H813F000
D |
| PR_EMS_AB_SUPPORTING_STACK_O | &H813E000
D |
| PR_EMS_AB_T_SELECTOR | &H8140010
2 |
| PR_EMS_AB_T_SELECTOR_INBOUND | &H8141010
2 |
| PR_EMS_AB_TAGGED_X509_CERT | &H8C6A110
2 |
| PR_EMS_AB_TARGET_ADDRESS | &H8011001
E |
| PR_EMS_AB_TARGET_MTAS | &H8142101
E |
| PR_EMS_AB_TELEPHONE_NUMBER | &H8012101
E |
| PR_EMS_AB_TELEPHONE_PERSONAL_PAGER | &H8C68001
E |
| PR_EMS_AB_TELETEX_TERMINAL_IDENTIFIER | &H8143110
2 |
| PR_EMS_AB_TEMP_ASSOC_THRESHOLD | &H8144000
3 |
| PR_EMS_AB_TOMBSTONE_LIFETIME | &H8145000
3 |
| PR_EMS_AB_TRACKING_LOG_PATH_NAME | &H8146001
E |
| PR_EMS_AB_TRANS_RETRY_MINS | &H8147000
3 |
| PR_EMS_AB_TRANS_TIMEOUT_MINS | &H8148000
3 |
| PR_EMS_AB_TRANSFER_RETRY_INTERVAL | &H8149000
3 |
| PR_EMS_AB_TRANSFER_TIMEOUT_NON_URGENT | &H814A000
3 |
| PR_EMS_AB_TRANSFER_TIMEOUT_NORMAL | &H814B000
3 |
| PR_EMS_AB_TRANSFER_TIMEOUT_URGENT | &H814C000
3 |
| PR_EMS_AB_TRANSLATION_TABLE_USED | &H814D000
3 |
| PR_EMS_AB_TRANSPORT_EXPEDITED_DATA | &H814E000
B |
| PR_EMS_AB_TRUST_LEVEL | &H814F000
3 |

| | |
|--------------------------------------|----------------|
| PR_EMS_AB_TURN_REQUEST_THRESHOLD | &H8150000
3 |
| PR_EMS_AB_TWO_WAY_ALTERNATE_FACILITY | &H8151000
B |
| PR_EMS_AB_TYPE | &H8C38001
E |
| PR_EMS_AB_UNAUTH_ORIG_BL | &H8152101
E |
| PR_EMS_AB_UNAUTH_ORIG_BL_O | &H8152000
D |
| PR_EMS_AB_USE_SERVER_VALUES | &H817E000
B |
| PR_EMS_AB_USE_SITE_VALUES | &H8183000
B |
| PR_EMS_AB_USENET_SITE_NAME | &H8189001
E |
| PR_EMS_AB_USER_PASSWORD | &H8153110
2 |
| PR_EMS_AB_USN_CHANGED | &H8029000
3 |
| PR_EMS_AB_USN_CREATED | &H8154000
3 |
| PR_EMS_AB_USN_DSA_LAST_OBJ_REMOVED | &H8155000
3 |
| PR_EMS_AB_USN_INTERSITE | &H817A000
3 |
| PR_EMS_AB_USN_LAST_OBJ_REM | &H8156000
3 |
| PR_EMS_AB_USN_SOURCE | &H8157000
3 |
| PR_EMS_AB_VIEW_CONTAINER_1 | &H81BF001
E |
| PR_EMS_AB_VIEW_CONTAINER_2 | &H81C0001
E |
| PR_EMS_AB_VIEW_CONTAINER_3 | &H81C1001
E |
| PR_EMS_AB_VIEW_DEFINITION | &H8C1B110
2 |
| PR_EMS_AB_VIEW_FLAGS | &H8C18000
3 |
| PR_EMS_AB_VIEW_SITE | &H81BE001
E |
| PR_EMS_AB_VOICE_MAIL_FLAGS | &H8C40110
2 |
| PR_EMS_AB_VOICE_MAIL_GREETINGS | &H8C3F101
E |
| PR_EMS_AB_VOICE_MAIL_PASSWORD | &H8C3D001
E |
| PR_EMS_AB_VOICE_MAIL_RECORDED_NAME | &H8C3E010 |

| | |
|--|-----------|
| | 2 |
| PR_EMS_AB_VOICE_MAIL_RECORDING_LENGTH | &H8C43100 |
| | 3 |
| PR_EMS_AB_VOICE_MAIL_SPEED | &H8C42000 |
| | 3 |
| PR_EMS_AB_VOICE_MAIL_SYSTEM_GUID | &H8C3B010 |
| | 2 |
| PR_EMS_AB_VOICE_MAIL_USER_ID | &H8C3C001 |
| | E |
| PR_EMS_AB_VOICE_MAIL_VOLUME | &H8C41000 |
| | 3 |
| PR_EMS_AB_WWW_HOME_PAGE | &H8175001 |
| | E |
| PR_EMS_AB_X121_ADDRESS | &H8158101 |
| | E |
| PR_EMS_AB_X25_CALL_USER_DATA_INCOMING | &H8159010 |
| | 2 |
| PR_EMS_AB_X25_CALL_USER_DATA_OUTGOING | &H815A010 |
| | 2 |
| PR_EMS_AB_X25_FACILITIES_DATA_INCOMING | &H815B010 |
| | 2 |
| PR_EMS_AB_X25_FACILITIES_DATA_OUTGOING | &H815C010 |
| | 2 |
| PR_EMS_AB_X25_LEASED_LINE_PORT | &H815D010 |
| | 2 |
| PR_EMS_AB_X25_LEASED_OR_SWITCHED | &H815E000 |
| | B |
| PR_EMS_AB_X25_REMOTE_MTA_PHONE | &H815F001 |
| | E |
| PR_EMS_AB_X400_ATTACHMENT_TYPE | &H8160010 |
| | 2 |
| PR_EMS_AB_X400_SELECTOR_SYNTAX | &H8161000 |
| | 3 |
| PR_EMS_AB_X500_ACCESS_CONTROL_LIST | &H8162010 |
| | 2 |
| PR_EMS_AB_X500_NC | &H81A2001 |
| | E |
| PR_EMS_AB_X500_RDN | &H81A1001 |
| | E |
| PR_EMS_AB_XMIT_TIMEOUT_NON_URGENT | &H8163000 |
| | 3 |
| PR_EMS_AB_XMIT_TIMEOUT_NORMAL | &H8164000 |
| | 3 |
| PR_EMS_AB_XMIT_TIMEOUT_URGENT | &H8165000 |
| | 3 |
| PR_EVENTS_ROOT_FOLDER_ENTRYID | &H668A010 |
| | 2 |
| PR_EXCESS_STORAGE_USED | &H3FF6000 |
| | 3 |

| | |
|-------------------------------------|----------------|
| PR_EXPIRY_NUMBER | &H3FED000
3 |
| PR_EXPIRY_UNITS | &H3FEE000
3 |
| PR_EXTENDED_ACL_DATA | &H3FFE010
2 |
| PR_FAST_TRANSFER | &H662F000
D |
| PR_FAVORITES_DEFAULT_NAME | &H6635001
E |
| PR_FOLDER_CHILD_COUNT | &H6638000
3 |
| PR_FOLDER_DESIGN_FLAGS | &H3FE2000
3 |
| PR_FOLDER_FLAGS | &H66A8000
3 |
| PR_FOLDER_PATHNAME | &H66B5001
E |
| PR_FOREIGN_ID | &H6667010
2 |
| PR_FOREIGN_REPORT_ID | &H6668010
2 |
| PR_FOREIGN_SUBJECT_ID | &H6669010
2 |
| PR_FREE_BUSY_FOR_LOCAL_SITE_ENTRYID | &H6625010
2 |
| PR_GW_ADMIN_OPERATIONS | &H6658000
3 |
| PR_GW_MTSIN_ENTRYID | &H6628010
2 |
| PR_GW_MTSOUT_ENTRYID | &H6629010
2 |
| PR_HAS_DAMS | &H3FEA000
B |
| PR_HAS_MODERATOR_RULES | &H663F000
B |
| PR_HAS_NAMED_PROPERTIES | &H664A000
B |
| PR_HAS_RULES | &H663A000
B |
| PR_HIERARCHY_CHANGE_NUM | &H663E000
3 |
| PR_HIERARCHY_SERVER | &H6623001
E |
| PR_HIERARCHY_SYNCHRONIZER | &H662C000
D |
| PR_IMAP_INTERNAL_DATE | &H65F5004
0 |
| PR_IN_CONFLICT | &H666C000 |

| | |
|--------------------------------|----------------|
| PR_IN_TRANSIT | B
&H6618000 |
| PR_INBOUND_NEWSFEED_DN | B
&H668D001 |
| PR_INTERNAL_TRACE_INFO | E
&H666A010 |
| PR_INTERNET_CHARSET | 2
&H669A001 |
| PR_INTERNET_CPID | E
&H3FDE000 |
| PR_INTERNET_NEWSGROUP_NAME | 3
&H66A7001 |
| PR_IPM_DAF_ENTRYID | E
&H661F010 |
| PR_IPM_FAVORITES_ENTRYID | 2
&H6630010 |
| PR_IPM_PUBLIC_FOLDERS_ENTRYID | 2
&H6631010 |
| PR_IS_NEWSGROUP | 2
&H6697000 |
| PR_IS_NEWSGROUP_ANCHOR | B
&H6696000 |
| PR_LAST_ACCESS_TIME | B
&H66A9004 |
| PR_LAST_FULL_BACKUP | 0
&H6685004 |
| PR_LAST_LOGOFF_TIME | 0
&H66A3004 |
| PR_LAST_LOGON_TIME | 0
&H66A2004 |
| PR_LAST_MODIFIER_ENTRYID | 0
&H3FFB010 |
| PR_LAST_MODIFIER_NAME | 2
&H3FFA001 |
| PR_LOCALE_ID | E
&H66A1000 |
| PR_LONGTERM_ENTRYID_FROM_TABLE | 3
&H6670010 |
| PR_MAILBOX_OWNER_ENTRYID | 2
&H661B010 |
| PR_MAILBOX_OWNER_NAME | 2
&H661C001 |
| PR_MEMBER_ENTRYID | E
&H0FFF010 |
| PR_MEMBER_ID | 2
&H6671001 |
| PR_MEMBER_NAME | 4
&H6672001 |
| | E |

| | |
|----------------------------------|----------------|
| PR_MEMBER_RIGHTS | &H6673000
3 |
| PR_MESSAGE_CODEPAGE | &H3FFD000
3 |
| PR_MESSAGE_LOCALE_ID | &H3FF1000
3 |
| PR_MESSAGE_PROCESSED | &H65E8000
B |
| PR_MESSAGE_SITE_NAME | &H65E7001
E |
| PR_MESSAGE_SIZE_EXTENDED | &H0E08001
4 |
| PR_MOVE_TO_FOLDER_ENTRYID | &H3FF4010
2 |
| PR_MOVE_TO_STORE_ENTRYID | &H3FF3010
2 |
| PR_MSG_BODY_ID | &H3FDD000
3 |
| PR_MTS_ID | &H0047010
2 |
| PR_MTS_REPORT_ID | &H0047010
2 |
| PR_MTS_SUBJECT_ID | &H6663010
2 |
| PR_NEW_SUBS_GET_AUTO_ADD | &H65E6000
B |
| PR_NEWSFEED_INFO | &H66A6010
2 |
| PR_NEWSGROUP_COMPONENT | &H66A5001
E |
| PR_NEWSGROUP_ROOT_FOLDER_ENTRYID | &H668C010
2 |
| PR_NNTP_ARTICLE_FOLDER_ENTRYID | &H668A010
2 |
| PR_NNTP_CONTROL_FOLDER_ENTRYID | &H668B010
2 |
| PR_NON_IPM_SUBTREE_ENTRYID | &H6620010
2 |
| PR_NORMAL_MESSAGE_SIZE | &H66B3000
3 |
| PR_NORMAL_MESSAGE_SIZE_EXTENDED | &H66B3001
4 |
| PR_NORMAL_MSG_W_ATTACH_COUNT | &H66AD000
3 |
| PR_NT_USER_NAME | &H66A0001
E |
| PR_OFFLINE_ADDRBOOK_ENTRYID | &H6623010
2 |
| PR_OFFLINE_FLAGS | &H663D000 |

| | |
|--------------------------------|-----------|
| | 3 |
| PR_OFFLINE_MESSAGE_ENTRYID | &H6627010 |
| | 2 |
| PR_OLDEST_DELETED_ON | &H6642004 |
| | 0 |
| PR_OOF_STATE | &H661D000 |
| | B |
| PR_ORIGINATOR_ADDR | &H665C001 |
| | E |
| PR_ORIGINATOR_ADDRTYPE | &H665D001 |
| | E |
| PR_ORIGINATOR_ENTRYID | &H665E010 |
| | 2 |
| PR_ORIGINATOR_NAME | &H665B001 |
| | E |
| PR_OST_ENCRYPTION | &H6702000 |
| | 3 |
| PR_OUTBOUND_NEWSFEED_DN | &H668E001 |
| | E |
| PR_OVERALL_AGE_LIMIT | &H6699000 |
| | 3 |
| PR_OVERALL_MSG_AGE_LIMIT | &H6693000 |
| | 3 |
| PR_OWA_URL | &H10F1001 |
| | E |
| PR_OWNER_COUNT | &H66B6000 |
| | 3 |
| PR_P1_CONTENT | &H1100010 |
| | 2 |
| PR_P1_CONTENT_TYPE | &H1101010 |
| | 2 |
| PR_PARENT_SOURCE_KEY | &H65E1010 |
| | 2 |
| PR_PREDECESSOR_CHANGE_LIST | &H65E3010 |
| | 2 |
| PR_PREVENT_MSG_CREATE | &H65F4000 |
| | B |
| PR_PREVIEW | &H3FD9001 |
| | E |
| PR_PREVIEW_UNREAD | &H3FD8001 |
| | E |
| PR_PROFILE_AB_FILES_PATH | &H660E001 |
| | E |
| PR_PROFILE_ADDR_INFO | &H6687010 |
| | 2 |
| PR_PROFILE_ALLPUB_COMMENT | &H6617001 |
| | E |
| PR_PROFILE_ALLPUB_DISPLAY_NAME | &H6616001 |
| | E |

| | |
|--------------------------------|----------------|
| PR_PROFILE_AUTH_PACKAGE | &H6619000
3 |
| PR_PROFILE_BINDING_ORDER | &H6609001
E |
| PR_PROFILE_CONFIG_FLAGS | &H6601000
3 |
| PR_PROFILE_CONNECT_FLAGS | &H6604000
3 |
| PR_PROFILE_FAVFLD_COMMENT | &H6615001
E |
| PR_PROFILE_FAVFLD_DISPLAY_NAME | &H660F001
E |
| PR_PROFILE_HOME_SERVER | &H6602001
E |
| PR_PROFILE_HOME_SERVER_ADDRS | &H6613101
E |
| PR_PROFILE_HOME_SERVER_DN | &H6612001
E |
| PR_PROFILE_MAILBOX | &H660B001
E |
| PR_PROFILE_MAX_RESTRICT | &H660D000
3 |
| PR_PROFILE_MOAB | &H667B001
E |
| PR_PROFILE_MOAB_GUID | &H667C001
E |
| PR_PROFILE_MOAB_SEQ | &H667D000
3 |
| PR_PROFILE_OFFLINE_INFO | &H6611010
2 |
| PR_PROFILE_OFFLINE_STORE_PATH | &H6610001
E |
| PR_PROFILE_OPEN_FLAGS | &H6609000
3 |
| PR_PROFILE_OPTIONS_DATA | &H6689010
2 |
| PR_PROFILE_SECURE_MAILBOX | &H67F0010
2 |
| PR_PROFILE_SERVER | &H660C001
E |
| PR_PROFILE_SERVER_DN | &H6614001
E |
| PR_PROFILE_TRANSPORT_FLAGS | &H6605000
3 |
| PR_PROFILE_TYPE | &H660A000
3 |
| PR_PROFILE_UI_STATE | &H6606000
3 |
| PR_PROFILE_UNRESOLVED_NAME | &H6607001 |

| | |
|----------------------------------|-----------|
| | E |
| PR_PROFILE_UNRESOLVED_SERVER | &H6608001 |
| | E |
| PR_PROFILE_USER | &H6603001 |
| | E |
| PR_PROFILE_VERSION | &H6600000 |
| | 3 |
| PR_PROMOTE_PROP_ID_LIST | &H666A010 |
| | 2 |
| PR_PST_ENCRYPTION | &H6702000 |
| | 3 |
| PR_PST_PATH | &H6700001 |
| | E |
| PR_PST_PW_SZ_NEW | &H6704001 |
| | E |
| PR_PST_PW_SZ_OLD | &H6703001 |
| | E |
| PR_PST_REMEMBER_PW | &H6701000 |
| | B |
| PR_PUBLIC_FOLDER_ENTRYID | &H663C010 |
| | 2 |
| PR_PUBLISH_IN_ADDRESS_BOOK | &H3FE6000 |
| | B |
| PR_RECIPIENT_NUMBER | &H6662000 |
| | 3 |
| PR_RECIPIENT_ON_ASSOC_MSG_COUNT | &H66B0000 |
| | 3 |
| PR_RECIPIENT_ON_NORMAL_MSG_COUNT | &H66AF000 |
| | 3 |
| PR_REPLICA_LIST | &H6698010 |
| | 2 |
| PR_REPLICA_SERVER | &H6644001 |
| | E |
| PR_REPLICA_VERSION | &H664B001 |
| | 4 |
| PR_REPLICATION_ALWAYS_INTERVAL | &H6694000 |
| | 3 |
| PR_REPLICATION_MESSAGE_PRIORITY | &H6692000 |
| | 3 |
| PR_REPLICATION_MSG_SIZE | &H6695000 |
| | 3 |
| PR_REPLICATION_SCHEDULE | &H6691010 |
| | 2 |
| PR_REPLICATION_STYLE | &H6690000 |
| | 3 |
| PR_REPLY_RECIPIENT_SMTP_PROXIES | &H3FFC001 |
| | E |
| PR_REPORT_DESTINATION_ENTRYID | &H6665010 |
| | 2 |

| | |
|----------------------------|----------------|
| PR_REPORT_DESTINATION_NAME | &H6664001
E |
| PR_RESOLVE_METHOD | &H3FE7000
3 |
| PR_RESTRICTION_COUNT | &H66AA000
3 |
| PR_RETENTION_AGE_LIMIT | &H66C4000
3 |
| PR_RIGHTS | &H6639000
3 |
| PR_RULE_ACTION_NUMBER | &H6650000
3 |
| PR_RULE_ACTION_TYPE | &H6649000
3 |
| PR_RULE_ACTIONS | &H668000F
E |
| PR_RULE_CONDITION | &H667900F
D |
| PR_RULE_ERROR | &H6648000
3 |
| PR_RULE_FOLDER_ENTRYID | &H6651010
2 |
| PR_RULE_ID | &H6674001
4 |
| PR_RULE_IDS | &H6675010
2 |
| PR_RULE_LEVEL | &H6683000
3 |
| PR_RULE_NAME | &H6682001
E |
| PR_RULE_PROVIDER | &H6681001
E |
| PR_RULE_PROVIDER_DATA | &H6684010
2 |
| PR_RULE_SEQUENCE | &H6676000
3 |
| PR_RULE_STATE | &H6677000
3 |
| PR_RULE_TRIGGER_HISTORY | &H3FF2010
2 |
| PR_RULE_USER_FLAGS | &H6678000
3 |
| PR_RULES_DATA | &H3FE1010
2 |
| PR_RULES_TABLE | &H3FE1000
D |
| PR_SCHEDULE_FOLDER_ENTRYID | &H661E010
2 |
| PR_SECURE_IN_SITE | &H669E000 |

| | |
|------------------------------|---------------------|
| PR_SECURE_ORIGINATION | B
&H3FE5000 |
| PR_SORT_LOCALE_ID | B
&H6705000
3 |
| PR_SOURCE_KEY | &H65E0010
2 |
| PR_SPLUS_FREE_BUSY_ENTRYID | &H6622010
2 |
| PR_STORAGE_LIMIT_INFORMATION | &H66A4000
3 |
| PR_STORAGE_QUOTA_LIMIT | &H3FF5000
3 |
| PR_STORE_OFFLINE | &H6632000
B |
| PR_STORE_SLOWLINK | &H7C0A000
B |
| PR_SUBJECT_TRACE_INFO | &H6661010
2 |
| PR_SVR_GENERATING_QUOTA_MSG | &H3FF7001
E |
| PR_SYNCHRONIZE_FLAGS | &H65E4000
3 |
| PR_SYS_CONFIG_FOLDER_ENTRYID | &H6636010
2 |
| PR_TEST_LINE_SPEED | &H662B010
2 |
| PR_TRACE_INFO | &H6660010
2 |
| PR_TRANSFER_ENABLED | &H662A000
B |
| PR_USER_ENTRYID | &H6619010
2 |
| PR_USER_NAME | &H661A001
E |
| PR_X400_ENVELOPE_TYPE | &H6653000
3 |

The following table lists the Microsoft Exchange property tags in numeric order:

| Hexadecimal value | Property tag (not in type library) |
|--------------------------|---|
| &H0047010
2 | PR_MTS_ID |
| &H0047010
2 | PR_MTS_REPORT_ID |
| &H0E08001
4 | PR_MESSAGE_SIZE_EXTENDED |
| &H0FFF010 | PR_MEMBER_ENTRYID |

2
&H10F1001 PR_OWA_URL
E
&H10F2000 PR_DISABLE_FULL_FIDELITY
B
&H1100010 PR_P1_CONTENT
2
&H1101010 PR_P1_CONTENT_TYPE
2
&H39FF001 PR_EMS_AB_DISPLAY_NAME_PRINTABLE
E
&H3FD8001 PR_PREVIEW_UNREAD
E
&H3FD9001 PR_PREVIEW
E
&H3FDA001 PR_ABSTRACT
E
&H3FDB000 PR_DL_REPORT_FLAGS
3
&H3FDC010 PR_BILATERAL_INFO
2
&H3FDD000 PR_MSG_BODY_ID
3
&H3FDE000 PR_INTERNET_CPID
3
&H3FDF000 PR_AUTO_RESPONSE_SUPPRESS
3
&H3FE0000 PR_ACL_TABLE
D
&H3FE0010 PR_ACL_DATA
2
&H3FE1000 PR_RULES_TABLE
D
&H3FE1010 PR_RULES_DATA
2
&H3FE2000 PR_FOLDER_DESIGN_FLAGS
3
&H3FE3000 PR_DELEGATED_BY_RULE
B
&H3FE4000 PR_DESIGN_IN_PROGRESS
B
&H3FE5000 PR_SECURE_ORIGINATION
B
&H3FE6000 PR_PUBLISH_IN_ADDRESS_BOOK
B
&H3FE7000 PR_RESOLVE_METHOD
3
&H3FE8001 PR_ADDRESS_BOOK_DISPLAY_NAME
E

&H3FE9000 PR_EFORMS_LOCALE_ID
3
&H3FEA000 PR_HAS_DAMS
B
&H3FEB000 PR_DEFERRED_SEND_NUMBER
3
&H3FEC000 PR_DEFERRED_SEND_UNITS
3
&H3FED000 PR_EXPIRY_NUMBER
3
&H3FEE000 PR_EXPIRY_UNITS
3
&H3FEF004 PR_DEFERRED_SEND_TIME
0
&H3FF0010 PR_CONFLICT_ENTRYID
2
&H3FF1000 PR_MESSAGE_LOCALE_ID
3
&H3FF2010 PR_RULE_TRIGGER_HISTORY
2
&H3FF3010 PR_MOVE_TO_STORE_ENTRYID
2
&H3FF4010 PR_MOVE_TO_FOLDER_ENTRYID
2
&H3FF5000 PR_STORAGE_QUOTA_LIMIT
3
&H3FF6000 PR_EXCESS_STORAGE_USED
3
&H3FF7001 PR_SVR_GENERATING_QUOTA_MSG
E
&H3FF8001 PR_CREATOR_NAME
E
&H3FF9010 PR_CREATOR_ENTRYID
2
&H3FFA001 PR_LAST_MODIFIER_NAME
E
&H3FFB010 PR_LAST_MODIFIER_ENTRYID
2
&H3FFC001 PR_REPLY_RECIPIENT_SMTP_PROXIES
E
&H3FFD000 PR_MESSAGE_CODEPAGE
3
&H3FFE010 PR_EXTENDED_ACL_DATA
2
&H65E0010 PR_SOURCE_KEY
2
&H65E1010 PR_PARENT_SOURCE_KEY
2
&H65E2010 PR_CHANGE_KEY

2
&H65E3010 PR_PREDECESSOR_CHANGE_LIST
2
&H65E4000 PR_SYNCHRONIZE_FLAGS
3
&H65E5000 PR_AUTO_ADD_NEW_SUBS
B
&H65E6000 PR_NEW_SUBS_GET_AUTO_ADD
B
&H65E7001 PR_MESSAGE_SITE_NAME
E
&H65E8000 PR_MESSAGE_PROCESSED
B
&H65F4000 PR_PREVENT_MSG_CREATE
B
&H65F5004 PR_IMAP_INTERNAL_DATE
0
&H6600000 PR_PROFILE_VERSION
3
&H6601000 PR_PROFILE_CONFIG_FLAGS
3
&H6602001 PR_PROFILE_HOME_SERVER
E
&H6603001 PR_PROFILE_USER
E
&H6604000 PR_PROFILE_CONNECT_FLAGS
3
&H6605000 PR_PROFILE_TRANSPORT_FLAGS
3
&H6606000 PR_PROFILE_UI_STATE
3
&H6607001 PR_PROFILE_UNRESOLVED_NAME
E
&H6608001 PR_PROFILE_UNRESOLVED_SERVER
E
&H6609000 PR_PROFILE_OPEN_FLAGS
3
&H6609001 PR_PROFILE_BINDING_ORDER
E
&H660A000 PR_PROFILE_TYPE
3
&H660B001 PR_PROFILE_MAILBOX
E
&H660C001 PR_PROFILE_SERVER
E
&H660D000 PR_PROFILE_MAX_RESTRICT
3
&H660E001 PR_PROFILE_AB_FILES_PATH
E

| | |
|----------------|--------------------------------|
| &H660F001
E | PR_PROFILE_FAVFLD_DISPLAY_NAME |
| &H6610001
E | PR_PROFILE_OFFLINE_STORE_PATH |
| &H6611010
2 | PR_PROFILE_OFFLINE_INFO |
| &H6612001
E | PR_PROFILE_HOME_SERVER_DN |
| &H6613101
E | PR_PROFILE_HOME_SERVER_ADDRS |
| &H6614001
E | PR_PROFILE_SERVER_DN |
| &H6615001
E | PR_PROFILE_FAVFLD_COMMENT |
| &H6616001
E | PR_PROFILE_ALLPUB_DISPLAY_NAME |
| &H6617001
E | PR_PROFILE_ALLPUB_COMMENT |
| &H6618000
3 | PR_DISABLE_WINSOCK |
| &H6618000
B | PR_IN_TRANSIT |
| &H6619000
3 | PR_PROFILE_AUTH_PACKAGE |
| &H6619010
2 | PR_USER_ENTRYID |
| &H661A001
E | PR_USER_NAME |
| &H661B010
2 | PR_MAILBOX_OWNER_ENTRYID |
| &H661C001
E | PR_MAILBOX_OWNER_NAME |
| &H661D000
B | PR_OOF_STATE |
| &H661E010
2 | PR_SCHEDULE_FOLDER_ENTRYID |
| &H661F010
2 | PR_IPM_DAF_ENTRYID |
| &H6620010
2 | PR_NON_IPM_SUBTREE_ENTRYID |
| &H6621010
2 | PR_EFORMS_REGISTRY_ENTRYID |
| &H6622010
2 | PR_SPLUS_FREE_BUSY_ENTRYID |
| &H6623001
E | PR_HIERARCHY_SERVER |
| &H6623010
2 | PR_OFFLINE_ADDRBOOK_ENTRYID |
| &H6624010 | PR_EFORMS_FOR_LOCALE_ENTRYID |

2
&H6625010 PR_FREE_BUSY_FOR_LOCAL_SITE_ENTRYID
2
&H6626010 PR_ADDRBOOK_FOR_LOCAL_SITE_ENTRYID
2
&H6627010 PR_OFFLINE_MESSAGE_ENTRYID
2
&H6628010 PR_GW_MTSIN_ENTRYID
2
&H6629010 PR_GW_MTSOUT_ENTRYID
2
&H662A000 PR_TRANSFER_ENABLED
B
&H662B010 PR_TEST_LINE_SPEED
2
&H662C000 PR_HIERARCHY_SYNCHRONIZER
D
&H662D000 PR_CONTENTS_SYNCHRONIZER
D
&H662E000 PR_COLLECTOR
D
&H662F000 PR_FAST_TRANSFER
D
&H6630010 PR_IPM_FAVORITES_ENTRYID
2
&H6631010 PR_IPM_PUBLIC_FOLDERS_ENTRYID
2
&H6632000 PR_STORE_OFFLINE
B
&H6634000 PR_CHANGE_ADVISOR
D
&H6635001 PR_FAVORITES_DEFAULT_NAME
E
&H6636010 PR_SYS_CONFIG_FOLDER_ENTRYID
2
&H6637004 PR_CHANGE_NOTIFICATION_GUID
8
&H6638000 PR_FOLDER_CHILD_COUNT
3
&H6639000 PR_RIGHTS
3
&H663A000 PR_HAS_RULES
B
&H663B010 PR_ADDRESS_BOOK_ENTRYID
2
&H663C010 PR_PUBLIC_FOLDER_ENTRYID
2
&H663D000 PR_OFFLINE_FLAGS
3

| | |
|-----------|----------------------------|
| &H663E000 | PR_HIERARCHY_CHANGE_NUM |
| 3 | |
| &H663F000 | PR_HAS_MODERATOR_RULES |
| B | |
| &H6640000 | PR_DELETED_MSG_COUNT |
| 3 | |
| &H6641000 | PR_DELETED_FOLDER_COUNT |
| 3 | |
| &H6642004 | PR_OLDEST_DELETED_ON |
| 0 | |
| &H6643000 | PR_DELETED_ASSOC_MSG_COUNT |
| 3 | |
| &H6644001 | PR_REPLICA_SERVER |
| E | |
| &H6645010 | PR_CLIENT_ACTIONS |
| 2 | |
| &H6646010 | PR_DAM_ORIGINAL_ENTRYID |
| 2 | |
| &H6647000 | PR_DAM_BACK_PATCHED |
| B | |
| &H6648000 | PR_RULE_ERROR |
| 3 | |
| &H6649000 | PR_RULE_ACTION_TYPE |
| 3 | |
| &H664A000 | PR_HAS_NAMED_PROPERTIES |
| B | |
| &H664B001 | PR_REPLICA_VERSION |
| 4 | |
| &H6650000 | PR_RULE_ACTION_NUMBER |
| 3 | |
| &H6651010 | PR_RULE_FOLDER_ENTRYID |
| 2 | |
| &H6652010 | PR_ACTIVE_USER_ENTRYID |
| 2 | |
| &H6653000 | PR_X400_ENVELOPE_TYPE |
| 3 | |
| &H6658000 | PR_GW_ADMIN_OPERATIONS |
| 3 | |
| &H665B001 | PR_ORIGINATOR_NAME |
| E | |
| &H665C001 | PR_ORIGINATOR_ADDR |
| E | |
| &H665D001 | PR_ORIGINATOR_ADDRTYPE |
| E | |
| &H665E010 | PR_ORIGINATOR_ENTRYID |
| 2 | |
| &H665F004 | PR_ARRIVAL_TIME |
| 0 | |
| &H6660010 | PR_TRACE_INFO |

2
&H6661010 PR_SUBJECT_TRACE_INFO
2
&H6662000 PR_RECIPIENT_NUMBER
3
&H6663010 PR_MTS_SUBJECT_ID
2
&H6664001 PR_REPORT_DESTINATION_NAME
E
&H6665010 PR_REPORT_DESTINATION_ENTRYID
2
&H6666010 PR_CONTENT_SEARCH_KEY
2
&H6667010 PR_FOREIGN_ID
2
&H6668010 PR_FOREIGN_REPORT_ID
2
&H6669010 PR_FOREIGN_SUBJECT_ID
2
&H666A010 PR_INTERNAL_TRACE_INFO
2
&H666A010 PR_PROMOTE_PROP_ID_LIST
2
&H666C000 PR_IN_CONFLICT
B
&H6670010 PR_LONGTERM_ENTRYID_FROM_TABLE
2
&H6671001 PR_MEMBER_ID
4
&H6672001 PR_MEMBER_NAME
E
&H6673000 PR_MEMBER_RIGHTS
3
&H6674001 PR_RULE_ID
4
&H6675010 PR_RULE_IDS
2
&H6676000 PR_RULE_SEQUENCE
3
&H6677000 PR_RULE_STATE
3
&H6678000 PR_RULE_USER_FLAGS
3
&H667900F PR_RULE_CONDITION
D
&H667B001 PR_PROFILE_MOAB
E
&H667C001 PR_PROFILE_MOAB_GUID
E

| | |
|----------------|----------------------------------|
| &H667D000
3 | PR_PROFILE_MOAB_SEQ |
| &H668000F
E | PR_RULE_ACTIONS |
| &H6681001
E | PR_RULE_PROVIDER |
| &H6682001
E | PR_RULE_NAME |
| &H6683000
3 | PR_RULE_LEVEL |
| &H6684010
2 | PR_RULE_PROVIDER_DATA |
| &H6685004
0 | PR_LAST_FULL_BACKUP |
| &H6687010
2 | PR_PROFILE_ADDR_INFO |
| &H6689010
2 | PR_PROFILE_OPTIONS_DATA |
| &H668A010
2 | PR_EVENTS_ROOT_FOLDER_ENTRYID |
| &H668A010
2 | PR_NNTP_ARTICLE_FOLDER_ENTRYID |
| &H668B010
2 | PR_NNTP_CONTROL_FOLDER_ENTRYID |
| &H668C010
2 | PR_NEWSGROUP_ROOT_FOLDER_ENTRYID |
| &H668D001
E | PR_INBOUND_NEWSFEED_DN |
| &H668E001
E | PR_OUTBOUND_NEWSFEED_DN |
| &H668F004
0 | PR_DELETED_ON |
| &H6690000
3 | PR_REPLICATION_STYLE |
| &H6691010
2 | PR_REPLICATION_SCHEDULE |
| &H6692000
3 | PR_REPLICATION_MESSAGE_PRIORITY |
| &H6693000
3 | PR_OVERALL_MSG_AGE_LIMIT |
| &H6694000
3 | PR_REPLICATION_ALWAYS_INTERVAL |
| &H6695000
3 | PR_REPLICATION_MSG_SIZE |
| &H6696000
B | PR_IS_NEWSGROUP_ANCHOR |
| &H6697000
B | PR_IS_NEWSGROUP |
| &H6698010 | PR_REPLICA_LIST |

2
&H6699000 PR_OVERALL_AGE_LIMIT
3
&H669A001 PR_INTERNET_CHARSET
E
&H669B001 PR_DELETED_MESSAGE_SIZE_EXTENDED
4
&H669C001 PR_DELETED_NORMAL_MESSAGE_SIZE_EXTENDED
4
&H669D001 PR_DELETED_ASSOC_MESSAGE_SIZE_EXTENDED
4
&H669E000 PR_SECURE_IN_SITE
B
&H66A0001 PR_NT_USER_NAME
E
&H66A1000 PR_LOCALE_ID
3
&H66A2004 PR_LAST_LOGON_TIME
0
&H66A3004 PR_LAST_LOGOFF_TIME
0
&H66A4000 PR_STORAGE_LIMIT_INFORMATION
3
&H66A5001 PR_NEWSGROUP_COMPONENT
E
&H66A6010 PR_NEWSFEED_INFO
2
&H66A7001 PR_INTERNET_NEWSGROUP_NAME
E
&H66A8000 PR_FOLDER_FLAGS
3
&H66A9004 PR_LAST_ACCESS_TIME
0
&H66AA000 PR_RESTRICTION_COUNT
3
&H66AB000 PR_CATEG_COUNT
3
&H66AC000 PR_CACHED_COLUMN_COUNT
3
&H66AD000 PR_NORMAL_MSG_W_ATTACH_COUNT
3
&H66AE000 PR_ASSOC_MSG_W_ATTACH_COUNT
3
&H66AF000 PR_RECIPIENT_ON_NORMAL_MSG_COUNT
3
&H66B0000 PR_RECIPIENT_ON_ASSOC_MSG_COUNT
3
&H66B1000 PR_ATTACH_ON_NORMAL_MSG_COUNT
3

&H66B2000 PR_ATTACH_ON_ASSOC_MSG_COUNT
3
&H66B3000 PR_NORMAL_MESSAGE_SIZE
3
&H66B3001 PR_NORMAL_MESSAGE_SIZE_EXTENDED
4
&H66B4000 PR_ASSOC_MESSAGE_SIZE
3
&H66B4001 PR_ASSOC_MESSAGE_SIZE_EXTENDED
4
&H66B5001 PR_FOLDER_PATHNAME
E
&H66B6000 PR_OWNER_COUNT
3
&H66B7000 PR_CONTACT_COUNT
3
&H66C3000 PR_CODE_PAGE_ID
3
&H66C4000 PR_RETENTION_AGE_LIMIT
3
&H6700001 PR_PST_PATH
E
&H6701000 PR_PST_REMEMBER_PW
B
&H6702000 PR_OST_ENCRYPTION
3
&H6702000 PR_PST_ENCRYPTION
3
&H6703001 PR_PST_PW_SZ_OLD
E
&H6704001 PR_PST_PW_SZ_NEW
E
&H6705000 PR_SORT_LOCALE_ID
3
&H67F0010 PR_PROFILE_SECURE_MAILBOX
2
&H7C0A000 PR_STORE_SLOWLINK
B
&H8001000 PR_EMS_AB_DISPLAY_NAME_OVERRIDE
B
&H8003110 PR_EMS_AB_CA_CERTIFICATE
2
&H8004001 PR_EMS_AB_FOLDER_PATHNAME
E
&H8005000 PR_EMS_AB_MANAGER
D
&H8005001 PR_EMS_AB_MANAGER_T
E
&H8006000 PR_EMS_AB_HOME_MDB_O

D
&H8006001 PR_EMS_AB_HOME_MDB
E
&H8007000 PR_EMS_AB_HOME_MTA_O
D
&H8007001 PR_EMS_AB_HOME_MTA
E
&H8008000 PR_EMS_AB_IS_MEMBER_OF_DL
D
&H8008001 PR_EMS_AB_IS_MEMBER_OF_DL_T
E
&H8009000 PR_EMS_AB_MEMBER
D
&H8009001 PR_EMS_AB_MEMBER_T
E
&H800A001 PR_EMS_AB_AUTOREPLY_MESSAGE
E
&H800B000 PR_EMS_AB_AUTOREPLY
B
&H800C000 PR_EMS_AB_OWNER_O
D
&H800C001 PR_EMS_AB_OWNER
E
&H800D000 PR_EMS_AB_KM_SERVER_O
D
&H800D001 PR_EMS_AB_KM_SERVER
E
&H800E000 PR_EMS_AB_REPORTS
D
&H800E001 PR_EMS_AB_REPORTS_T
D
&H800F101 PR_EMS_AB_PROXY_ADDRESSES
E
&H8010010 PR_EMS_AB_HELP_DATA32
2
&H8011001 PR_EMS_AB_TARGET_ADDRESS
E
&H8012101 PR_EMS_AB_TELEPHONE_NUMBER
E
&H8013010 PR_EMS_AB_NT_SECURITY_DESCRIPTOR
2
&H8014000 PR_EMS_AB_HOME_MDB_BL_O
D
&H8014101 PR_EMS_AB_HOME_MDB_BL
E
&H8015000 PR_EMS_AB_PUBLIC_DELEGATES
D
&H8015001 PR_EMS_AB_PUBLIC_DELEGATES_T
E

| | |
|----------------|---------------------------------------|
| &H8016010
2 | PR_EMS_AB_CERTIFICATE_REVOCATION_LIST |
| &H8017010
2 | PR_EMS_AB_ADDRESS_ENTRY_DISPLAY_TABLE |
| &H8018010
2 | PR_EMS_AB_ADDRESS_SYNTAX |
| &H8023010
2 | PR_EMS_AB_BUSINESS_ROLES |
| &H8024000
D | PR_EMS_AB_OWNER_BL_O |
| &H8024101
E | PR_EMS_AB_OWNER_BL |
| &H8025110
2 | PR_EMS_AB_CROSS_CERTIFICATE_PAIR |
| &H8026110
2 | PR_EMS_AB_AUTHORITY_REVOCATION_LIST |
| &H8027010
2 | PR_EMS_AB_ASSOC_NT_ACCOUNT |
| &H8028004
0 | PR_EMS_AB_EXPIRATION_TIME |
| &H8029000
3 | PR_EMS_AB_USN_CHANGED |
| &H802D001
E | PR_EMS_AB_EXTENSION_ATTRIBUTE_1 |
| &H802E001
E | PR_EMS_AB_EXTENSION_ATTRIBUTE_2 |
| &H802F001
E | PR_EMS_AB_EXTENSION_ATTRIBUTE_3 |
| &H8030001
E | PR_EMS_AB_EXTENSION_ATTRIBUTE_4 |
| &H8031001
E | PR_EMS_AB_EXTENSION_ATTRIBUTE_5 |
| &H8032001
E | PR_EMS_AB_EXTENSION_ATTRIBUTE_6 |
| &H8033001
E | PR_EMS_AB_EXTENSION_ATTRIBUTE_7 |
| &H8034001
E | PR_EMS_AB_EXTENSION_ATTRIBUTE_8 |
| &H8035001
E | PR_EMS_AB_EXTENSION_ATTRIBUTE_9 |
| &H8036001
E | PR_EMS_AB_EXTENSION_ATTRIBUTE_10 |
| &H8037110
2 | PR_EMS_AB_SECURITY_PROTOCOL |
| &H8038000
D | PR_EMS_AB_PF_CONTACTS_O |
| &H8038101
E | PR_EMS_AB_PF_CONTACTS |
| &H803A010 | PR_EMS_AB_HELP_DATA16 |

2
&H803B001 PR_EMS_AB_HELP_FILE_NAME
E
&H803C000 PR_EMS_AB_OBJ_DIST_NAME_O
D
&H803C001 PR_EMS_AB_OBJ_DIST_NAME
E
&H803D001 PR_EMS_AB_ENCRYPT_ALG_SELECTED_OTHER
E
&H803E001 PR_EMS_AB_AUTOREPLY_SUBJECT
E
&H803F000 PR_EMS_AB_HOME_PUBLIC_SERVER_O
D
&H803F001 PR_EMS_AB_HOME_PUBLIC_SERVER
E
&H8040101 PR_EMS_AB_ENCRYPT_ALG_LIST_NA
E
&H8041101 PR_EMS_AB_ENCRYPT_ALG_LIST_OTHER
E
&H8042001 PR_EMS_AB_IMPORTED_FROM
E
&H8043001 PR_EMS_AB_ENCRYPT_ALG_SELECTED_NA
E
&H8044000 PR_EMS_AB_ACCESS_CATEGORY
3
&H8045010 PR_EMS_AB_ACTIVATION_SCHEDULE
2
&H8046000 PR_EMS_AB_ACTIVATION_STYLE
3
&H8047010 PR_EMS_AB_ADDRESS_ENTRY_DISPLAY_TABLE_MSDO
2 S
&H8048001 PR_EMS_AB_ADDRESS_TYPE
E
&H8049001 PR_EMS_AB_ADMD
E
&H804A001 PR_EMS_AB_ADMIN_DESCRIPTION
E
&H804B001 PR_EMS_AB_ADMIN_DISPLAY_NAME
E
&H804C001 PR_EMS_AB_ADMIN_EXTENSION_DLL
E
&H804D000 PR_EMS_AB_ALIASED_OBJECT_NAME_O
D
&H804D001 PR_EMS_AB_ALIASED_OBJECT_NAME
E
&H804E000 PR_EMS_AB_ALT_RECIPIENT_O
D
&H804E001 PR_EMS_AB_ALT_RECIPIENT
E

| | |
|----------------|----------------------------------|
| &H804F000
D | PR_EMS_AB_ALT_RECIPIENT_BL_O |
| &H804F101
E | PR_EMS_AB_ALT_RECIPIENT_BL |
| &H8050010
2 | PR_EMS_AB_ANCESTOR_ID |
| &H8051000
D | PR_EMS_AB_ASSOC_REMOTE_DXA_O |
| &H8051101
E | PR_EMS_AB_ASSOC_REMOTE_DXA |
| &H8052000
3 | PR_EMS_AB_ASSOCIATION_LIFETIME |
| &H8053000
D | PR_EMS_AB_AUTH_ORIG_BL_O |
| &H8053101
E | PR_EMS_AB_AUTH_ORIG_BL |
| &H8054001
E | PR_EMS_AB_AUTHORIZED_DOMAIN |
| &H8055010
2 | PR_EMS_AB_AUTHORIZED_PASSWORD |
| &H8056001
E | PR_EMS_AB_AUTHORIZED_USER |
| &H8057101
E | PR_EMS_AB_BUSINESS_CATEGORY |
| &H8058000
D | PR_EMS_AB_CAN_CREATE_PF_O |
| &H8058101
E | PR_EMS_AB_CAN_CREATE_PF |
| &H8059000
D | PR_EMS_AB_CAN_CREATE_PF_BL_O |
| &H8059101
E | PR_EMS_AB_CAN_CREATE_PF_BL |
| &H805A000
D | PR_EMS_AB_CAN_CREATE_PF_DL_O |
| &H805A101
E | PR_EMS_AB_CAN_CREATE_PF_DL |
| &H805B000
D | PR_EMS_AB_CAN_CREATE_PF_DL_BL_O |
| &H805B101
E | PR_EMS_AB_CAN_CREATE_PF_DL_BL |
| &H805C000
D | PR_EMS_AB_CAN_NOT_CREATE_PF_O |
| &H805C101
E | PR_EMS_AB_CAN_NOT_CREATE_PF |
| &H805D000
D | PR_EMS_AB_CAN_NOT_CREATE_PF_BL_O |
| &H805D101
E | PR_EMS_AB_CAN_NOT_CREATE_PF_BL |
| &H805E000 | PR_EMS_AB_CAN_NOT_CREATE_PF_DL_O |

D
&H805E101 PR_EMS_AB_CAN_NOT_CREATE_PF_DL
E
&H805F000 PR_EMS_AB_CAN_NOT_CREATE_PF_DL_BL_O
D
&H805F101 PR_EMS_AB_CAN_NOT_CREATE_PF_DL_BL
E
&H8060000 PR_EMS_AB_CAN_PRESERVE_DNS
B
&H8061000 PR_EMS_AB_CLOCK_ALERT_OFFSET
3
&H8062000 PR_EMS_AB_CLOCK_ALERT_REPAIR
B
&H8063000 PR_EMS_AB_CLOCK_WARNING_OFFSET
3
&H8064000 PR_EMS_AB_CLOCK_WARNING_REPAIR
B
&H8065001 PR_EMS_AB_COMPUTER_NAME
E
&H8066101 PR_EMS_AB_CONNECTED_DOMAINS
E
&H8067000 PR_EMS_AB_CONTAINER_INFO
3
&H8068000 PR_EMS_AB_COST
3
&H8069001 PR_EMS_AB_COUNTRY_NAME
E
&H806A000 PR_EMS_AB_DELIV_CONT_LENGTH
3
&H806B110 PR_EMS_AB_DELIV_EITS
2
&H806C110 PR_EMS_AB_DELIV_EXT_CONT_TYPES
2
&H806D000 PR_EMS_AB_DELIVER_AND_REDIRECT
B
&H806E000 PR_EMS_AB_DELIVERY_MECHANISM
3
&H806F101 PR_EMS_AB_DESCRIPTION
E
&H8070101 PR_EMS_AB_DESTINATION_INDICATOR
E
&H8071001 PR_EMS_AB_DIAGNOSTIC_REG_KEY
E
&H8072000 PR_EMS_AB_DL_MEM_REJECT_PERMS_BL_O
D
&H8072101 PR_EMS_AB_DL_MEM_REJECT_PERMS_BL
E
&H8073000 PR_EMS_AB_DL_MEM_SUBMIT_PERMS_BL_O
D

&H8073101 PR_EMS_AB_DL_MEM_SUBMIT_PERMS_BL
E
&H8074110 PR_EMS_AB_DL_MEMBER_RULE
2
&H8075000 PR_EMS_AB_DOMAIN_DEF_ALT_RECIP_O
D
&H8075001 PR_EMS_AB_DOMAIN_DEF_ALT_RECIP
E
&H8076001 PR_EMS_AB_DOMAIN_NAME
E
&H8077010 PR_EMS_AB_DSA_SIGNATURE
2
&H8078000 PR_EMS_AB_DXA_ADMIN_COPY
B
&H8079000 PR_EMS_AB_DXA_ADMIN_FORWARD
B
&H807A000 PR_EMS_AB_DXA_ADMIN_UPDATE
3
&H807B000 PR_EMS_AB_DXA_APPEND_REQCN
B
&H807C000 PR_EMS_AB_DXA_CONF_CONTAINER_LIST_O
D
&H807C101 PR_EMS_AB_DXA_CONF_CONTAINER_LIST
E
&H807D004 PR_EMS_AB_DXA_CONF_REQ_TIME
0
&H807E001 PR_EMS_AB_DXA_CONF_SEQ
E
&H807F000 PR_EMS_AB_DXA_CONF_SEQ_USN
3
&H8080000 PR_EMS_AB_DXA_EXCHANGE_OPTIONS
3
&H8081000 PR_EMS_AB_DXA_EXPORT_NOW
B
&H8082000 PR_EMS_AB_DXA_FLAGS
3
&H8083001 PR_EMS_AB_DXA_IMP_SEQ
E
&H8084004 PR_EMS_AB_DXA_IMP_SEQ_TIME
0
&H8085000 PR_EMS_AB_DXA_IMP_SEQ_USN
3
&H8086000 PR_EMS_AB_DXA_IMPORT_NOW
B
&H8087101 PR_EMS_AB_DXA_IN_TEMPLATE_MAP
E
&H8088000 PR_EMS_AB_DXA_LOCAL_ADMIN_O
D
&H8088001 PR_EMS_AB_DXA_LOCAL_ADMIN

E
&H8089000 PR_EMS_AB_DXA_LOGGING_LEVEL
3
&H808A001 PR_EMS_AB_DXA_NATIVE_ADDRESS_TYPE
E
&H808B101 PR_EMS_AB_DXA_OUT_TEMPLATE_MAP
E
&H808C001 PR_EMS_AB_DXA_PASSWORD
E
&H808D000 PR_EMS_AB_DXA_PREV_EXCHANGE_OPTIONS
3
&H808E000 PR_EMS_AB_DXA_PREV_EXPORT_NATIVE_ONLY
B
&H808F000 PR_EMS_AB_DXA_PREV_IN_EXCHANGE_SENSITIVITY
3
&H8090000 PR_EMS_AB_DXA_PREV_REMOTE_ENTRIES_O
D
&H8090001 PR_EMS_AB_DXA_PREV_REMOTE_ENTRIES
E
&H8091000 PR_EMS_AB_DXA_PREV_REPLICATION_SENSITIVITY
3
&H8092000 PR_EMS_AB_DXA_PREV_TEMPLATE_OPTIONS
3
&H8093000 PR_EMS_AB_DXA_PREV_TYPES
3
&H8094001 PR_EMS_AB_DXA_RECIPIENT_CP
E
&H8095000 PR_EMS_AB_DXA_REMOTE_CLIENT_O
D
&H8095001 PR_EMS_AB_DXA_REMOTE_CLIENT
E
&H8096001 PR_EMS_AB_DXA_REQ_SEQ
E
&H8097004 PR_EMS_AB_DXA_REQ_SEQ_TIME
0
&H8098000 PR_EMS_AB_DXA_REQ_SEQ_USN
3
&H8099001 PR_EMS_AB_DXA_REQNAME
E
&H809A001 PR_EMS_AB_DXA_SVR_SEQ
E
&H809B004 PR_EMS_AB_DXA_SVR_SEQ_TIME
0
&H809C000 PR_EMS_AB_DXA_SVR_SEQ_USN
3
&H809D000 PR_EMS_AB_DXA_TASK
3
&H809E000 PR_EMS_AB_DXA_TEMPLATE_OPTIONS
3

&H809F004 PR_EMS_AB_DXA_TEMPLATE_TIMESTAMP
0
&H80A0000 PR_EMS_AB_DXA_TYPES
3
&H80A1000 PR_EMS_AB_DXA_UNCONF_CONTAINER_LIST_O
D
&H80A1101 PR_EMS_AB_DXA_UNCONF_CONTAINER_LIST
E
&H80A2000 PR_EMS_AB_ENCAPSULATION_METHOD
3
&H80A3000 PR_EMS_AB_ENCRYPT
B
&H80A4000 PR_EMS_AB_EXPAND_DLS_LOCALLY
B
&H80A5000 PR_EMS_AB_EXPORT_CONTAINERS_O
D
&H80A5101 PR_EMS_AB_EXPORT_CONTAINERS
E
&H80A6000 PR_EMS_AB_EXPORT_CUSTOM_RECIPIENTS
B
&H80A7000 PR_EMS_AB_EXTENDED_CHARS_ALLOWED
B
&H80A8110 PR_EMS_AB_EXTENSION_DATA
2
&H80A9101 PR_EMS_AB_EXTENSION_NAME
E
&H80AA101 PR_EMS_AB_EXTENSION_NAME_INHERITED
E
&H80AB110 PR_EMS_AB_FACSIMILE_TELEPHONE_NUMBER
2
&H80AC010 PR_EMS_AB_FILE_VERSION
2
&H80AD000 PR_EMS_AB_FILTER_LOCAL_ADDRESSES
B
&H80AE000 PR_EMS_AB_FOLDERS_CONTAINER_O
D
&H80AE001 PR_EMS_AB_FOLDERS_CONTAINER
E
&H80AF000 PR_EMS_AB_GARBAGE_COLL_PERIOD
3
&H80B0001 PR_EMS_AB_GATEWAY_LOCAL_CRED
E
&H80B1001 PR_EMS_AB_GATEWAY_LOCAL_DESIG
E
&H80B2101 PR_EMS_AB_GATEWAY_PROXY
E
&H80B3010 PR_EMS_AB_GATEWAY_ROUTING_TREE
2
&H80B4004 PR_EMS_AB_GWART_LAST_MODIFIED

0
&H80B5000 PR_EMS_AB_HAS_FULL_REPLICA_NCS_O
D
&H80B5101 PR_EMS_AB_HAS_FULL_REPLICA_NCS
E
&H80B6000 PR_EMS_AB_HAS_MASTER_NCS_O
D
&H80B6101 PR_EMS_AB_HAS_MASTER_NCS
E
&H80B7000 PR_EMS_AB_HEURISTICS
3
&H80B8000 PR_EMS_AB_HIDE_DL_MEMBERSHIP
B
&H80B9000 PR_EMS_AB_HIDE_FROM_ADDRESS_BOOK
B
&H80BA000 PR_EMS_AB_IMPORT_CONTAINER_O
D
&H80BA001 PR_EMS_AB_IMPORT_CONTAINER
E
&H80BB000 PR_EMS_AB_IMPORT_SENSITIVITY
3
&H80BC000 PR_EMS_AB_INBOUND_SITES_O
D
&H80BC101 PR_EMS_AB_INBOUND_SITES
E
&H80BD000 PR_EMS_AB_INSTANCE_TYPE
3
&H80BE101 PR_EMS_AB_INTERNATIONAL_ISDN_NUMBER
E
&H80BF010 PR_EMS_AB_INVOCATION_ID
2
&H80C0000 PR_EMS_AB_IS_DELETED
B
&H80C1000 PR_EMS_AB_IS_SINGLE_VALUED
B
&H80C2110 PR_EMS_AB_KCC_STATUS
2
&H80C3101 PR_EMS_AB_KNOWLEDGE_INFORMATION
E
&H80C4000 PR_EMS_AB_LINE_WRAP
3
&H80C5000 PR_EMS_AB_LINK_ID
3
&H80C6001 PR_EMS_AB_LOCAL_BRIDGE_HEAD
E
&H80C7001 PR_EMS_AB_LOCAL_BRIDGE_HEAD_ADDRESS
E
&H80C8000 PR_EMS_AB_LOCAL_INITIAL_TURN
B

&H80C9000 PR_EMS_AB_LOCAL_SCOPE_O
D
&H80C9101 PR_EMS_AB_LOCAL_SCOPE
E
&H80CA001 PR_EMS_AB_LOG_FILENAME
E
&H80CB000 PR_EMS_AB_LOG_ROLLOVER_INTERVAL
3
&H80CC000 PR_EMS_AB_MAINTAIN_AUTOREPLY_HISTORY
B
&H80CD000 PR_EMS_AB_MAPI_DISPLAY_TYPE
3
&H80CE000 PR_EMS_AB_MAPI_ID
3
&H80CF000 PR_EMS_AB_MDB_BACKOFF_INTERVAL
3
&H80D0000 PR_EMS_AB_MDB_MSG_TIME_OUT_PERIOD
3
&H80D1000 PR_EMS_AB_MDB_OVER_QUOTA_LIMIT
3
&H80D2000 PR_EMS_AB_MDB_STORAGE_QUOTA
3
&H80D3000 PR_EMS_AB_MDB_UNREAD_LIMIT
3
&H80D4000 PR_EMS_AB_MDB_USE_DEFAULTS
B
&H80D5000 PR_EMS_AB_MESSAGE_TRACKING_ENABLED
B
&H80D6000 PR_EMS_AB_MONITOR_CLOCK
B
&H80D7000 PR_EMS_AB_MONITOR_SERVERS
B
&H80D8000 PR_EMS_AB_MONITOR_SERVICES
B
&H80D9000 PR_EMS_AB_MONITORED_CONFIGURATIONS_O
D
&H80D9101 PR_EMS_AB_MONITORED_CONFIGURATIONS
E
&H80DA000 PR_EMS_AB_MONITORED_SERVERS_O
D
&H80DA101 PR_EMS_AB_MONITORED_SERVERS
E
&H80DB101 PR_EMS_AB_MONITORED_SERVICES
E
&H80DC000 PR_EMS_AB_MONITORING_ALERT_DELAY
3
&H80DD000 PR_EMS_AB_MONITORING_ALERT_UNITS
3
&H80DE000 PR_EMS_AB_MONITORING_AVAILABILITY_STYLE

3
&H80DF010 PR_EMS_AB_MONITORING_AVAILABILITY_WINDOW
2
&H80E0000 PR_EMS_AB_MONITORING_CACHED_VIA_MAIL_O
D
&H80E0101 PR_EMS_AB_MONITORING_CACHED_VIA_MAIL
E
&H80E1000 PR_EMS_AB_MONITORING_CACHED_VIA_RPC_O
D
&H80E1101 PR_EMS_AB_MONITORING_CACHED_VIA_RPC
E
&H80E2110 PR_EMS_AB_MONITORING_ESCALATION_PROCEDURE
2
&H80E3000 PR_EMS_AB_MONITORING_HOTSITE_POLL_INTERVAL
3
&H80E4000 PR_EMS_AB_MONITORING_HOTSITE_POLL_UNITS
3
&H80E5000 PR_EMS_AB_MONITORING_MAIL_UPDATE_INTERVAL
3
&H80E6000 PR_EMS_AB_MONITORING_MAIL_UPDATE_UNITS
3
&H80E7000 PR_EMS_AB_MONITORING_NORMAL_POLL_INTERVAL
3
&H80E8000 PR_EMS_AB_MONITORING_NORMAL_POLL_UNITS
3
&H80E9000 PR_EMS_AB_MONITORING_RECIPIENTS_O
D
&H80E9101 PR_EMS_AB_MONITORING_RECIPIENTS
E
&H80EA000 PR_EMS_AB_MONITORING_RECIPIENTS_NDR_O
D
&H80EA101 PR_EMS_AB_MONITORING_RECIPIENTS_NDR
E
&H80EB000 PR_EMS_AB_MONITORING_RPC_UPDATE_INTERVAL
3
&H80EC000 PR_EMS_AB_MONITORING_RPC_UPDATE_UNITS
3
&H80ED000 PR_EMS_AB_MONITORING_WARNING_DELAY
3
&H80EE000 PR_EMS_AB_MONITORING_WARNING_UNITS
3
&H80EF001 PR_EMS_AB_MTA_LOCAL_CRED
E
&H80F0001 PR_EMS_AB_MTA_LOCAL_DESIG
E
&H80F1010 PR_EMS_AB_N_ADDRESS
2
&H80F2000 PR_EMS_AB_N_ADDRESS_TYPE
3

| | |
|----------------|--|
| &H80F3001
E | PR_EMS_AB_NT_MACHINE_NAME |
| &H80F4000
3 | PR_EMS_AB_NUM_OF_OPEN_RETRIES |
| &H80F5000
3 | PR_EMS_AB_NUM_OF_TRANSFER_RETRIES |
| &H80F6000
3 | PR_EMS_AB_OBJECT_CLASS_CATEGORY |
| &H80F7000
3 | PR_EMS_AB_OBJECT_VERSION |
| &H80F8000
D | PR_EMS_AB_OFF_LINE_AB_CONTAINERS_O |
| &H80F8101
E | PR_EMS_AB_OFF_LINE_AB_CONTAINERS |
| &H80F9010
2 | PR_EMS_AB_OFF_LINE_AB_SCHEDULE |
| &H80FA000
D | PR_EMS_AB_OFF_LINE_AB_SERVER_O |
| &H80FA001
E | PR_EMS_AB_OFF_LINE_AB_SERVER |
| &H80FB000
3 | PR_EMS_AB_OFF_LINE_AB_STYLE |
| &H80FC000
3 | PR_EMS_AB_OID_TYPE |
| &H80FD010
2 | PR_EMS_AB_OM_OBJECT_CLASS |
| &H80FE000
3 | PR_EMS_AB_OM_SYNTAX |
| &H80FF000
B | PR_EMS_AB_OOF_REPLY_TO_ORIGINATOR |
| &H8100000
3 | PR_EMS_AB_OPEN_RETRY_INTERVAL |
| &H8101101
E | PR_EMS_AB_ORGANIZATION_NAME |
| &H8102101
E | PR_EMS_AB_ORGANIZATIONAL_UNIT_NAME |
| &H8103010
2 | PR_EMS_AB_ORIGINAL_DISPLAY_TABLE |
| &H8104010
2 | PR_EMS_AB_ORIGINAL_DISPLAY_TABLE_MSDOS |
| &H8105000
D | PR_EMS_AB_OUTBOUND_SITES_O |
| &H8105101
E | PR_EMS_AB_OUTBOUND_SITES |
| &H8106010
2 | PR_EMS_AB_P_SELECTOR |
| &H8107010
2 | PR_EMS_AB_P_SELECTOR_INBOUND |
| &H8108010 | PR_EMS_AB_PER_MSG_DIALOG_DISPLAY_TABLE |

2
&H8109010 PR_EMS_AB_PER_RECIP_DIALOG_DISPLAY_TABLE
2
&H810A010 PR_EMS_AB_PERIOD_REP_SYNC_TIMES
2
&H810B000 PR_EMS_AB_PERIOD_REPL_STAGGER
3
&H810C110 PR_EMS_AB_POSTAL_ADDRESS
2
&H810D100 PR_EMS_AB_PREFERRED_DELIVERY_METHOD
3
&H810E001 PR_EMS_AB_PRMD
E
&H810F001 PR_EMS_AB_PROXY_GENERATOR_DLL
E
&H8110000 PR_EMS_AB_PUBLIC_DELEGATES_BL_O
D
&H8110101 PR_EMS_AB_PUBLIC_DELEGATES_BL
E
&H81110102 PR_EMS_AB_QUOTA_NOTIFICATION_SCHEDULE
&H8112000 PR_EMS_AB_QUOTA_NOTIFICATION_STYLE
3
&H8113000 PR_EMS_AB_RANGE_LOWER
3
&H8114000 PR_EMS_AB_RANGE_UPPER
3
&H8115001 PR_EMS_AB_RAS_CALLBACK_NUMBER
E
&H8116001 PR_EMS_AB_RAS_PHONE_NUMBER
E
&H8117001 PR_EMS_AB_RAS_PHONEBOOK_ENTRY_NAME
E
&H8118001 PR_EMS_AB_RAS_REMOTE_SRVR_NAME
E
&H81191102 PR_EMS_AB_REGISTERED_ADDRESS
&H811A001 PR_EMS_AB_REMOTE_BRIDGE_HEAD
E
&H811B001 PR_EMS_AB_REMOTE_BRIDGE_HEAD_ADDRESS
E
&H811C000 PR_EMS_AB_REMOTE_OUT_BH_SERVER_O
D
&H811C001 PR_EMS_AB_REMOTE_OUT_BH_SERVER
E
&H811D000 PR_EMS_AB_REMOTE_SITE_O
D
&H811D001 PR_EMS_AB_REMOTE_SITE
E
&H811E000 PR_EMS_AB_REPLICATION_SENSITIVITY
3

| | |
|----------------|-----------------------------------|
| &H811F000
3 | PR_EMS_AB_REPLICATION_STAGGER |
| &H8120000
B | PR_EMS_AB_REPORT_TO_ORIGINATOR |
| &H8121000
B | PR_EMS_AB_REPORT_TO_OWNER |
| &H8122000
3 | PR_EMS_AB_REQ_SEQ |
| &H8123000
D | PR_EMS_AB_RESPONSIBLE_LOCAL_DXA_O |
| &H8123001
E | PR_EMS_AB_RESPONSIBLE_LOCAL_DXA |
| &H8124000
D | PR_EMS_AB_RID_SERVER_O |
| &H8124001
E | PR_EMS_AB_RID_SERVER |
| &H8125000
D | PR_EMS_AB_ROLE_OCCUPANT_O |
| &H8125101
E | PR_EMS_AB_ROLE_OCCUPANT |
| &H8126101
E | PR_EMS_AB_ROUTING_LIST |
| &H8127000
3 | PR_EMS_AB_RTS_CHECKPOINT_SIZE |
| &H8128000
3 | PR_EMS_AB_RTS_RECOVERY_TIMEOUT |
| &H8129000
3 | PR_EMS_AB_RTS_WINDOW_SIZE |
| &H812A000
D | PR_EMS_AB_RUNS_ON_O |
| &H812A101
E | PR_EMS_AB_RUNS_ON |
| &H812B010
2 | PR_EMS_AB_S_SELECTOR |
| &H812C010
2 | PR_EMS_AB_S_SELECTOR_INBOUND |
| &H812D000
3 | PR_EMS_AB_SEARCH_FLAGS |
| &H812E110
2 | PR_EMS_AB_SEARCH_GUIDE |
| &H812F000
D | PR_EMS_AB_SEE_ALSO_O |
| &H812F101
E | PR_EMS_AB_SEE_ALSO |
| &H8130101
E | PR_EMS_AB_SERIAL_NUMBER |
| &H8131000
3 | PR_EMS_AB_SERVICE_ACTION_FIRST |
| &H8132000 | PR_EMS_AB_SERVICE_ACTION_OTHER |

3
&H8133000 PR_EMS_AB_SERVICE_ACTION_SECOND
3
&H8134000 PR_EMS_AB_SERVICE_RESTART_DELAY
3
&H8135001 PR_EMS_AB_SERVICE_RESTART_MESSAGE
E
&H8136000 PR_EMS_AB_SESSION_DISCONNECT_TIMER
3
&H8137101 PR_EMS_AB_SITE_AFFINITY
E
&H8138101 PR_EMS_AB_SITE_PROXY_SPACE
E
&H8139004 PR_EMS_AB_SPACE_LAST_COMPUTED
0
&H813A001 PR_EMS_AB_STREET_ADDRESS
E
&H813B000 PR_EMS_AB_SUB_REFS_O
D
&H813B101 PR_EMS_AB_SUB_REFS
E
&H813C000 PR_EMS_AB_SUBMISSION_CONT_LENGTH
3
&H813D110 PR_EMS_AB_SUPPORTED_APPLICATION_CONTEXT
2
&H813E000 PR_EMS_AB_SUPPORTING_STACK_O
D
&H813E101 PR_EMS_AB_SUPPORTING_STACK
E
&H813F000 PR_EMS_AB_SUPPORTING_STACK_BL_O
D
&H813F101 PR_EMS_AB_SUPPORTING_STACK_BL
E
&H8140010 PR_EMS_AB_T_SELECTOR
2
&H8141010 PR_EMS_AB_T_SELECTOR_INBOUND
2
&H8142101 PR_EMS_AB_TARGET_MMAS
E
&H8143110 PR_EMS_AB_TELETEX_TERMINAL_IDENTIFIER
2
&H8144000 PR_EMS_AB_TEMP_ASSOC_THRESHOLD
3
&H8145000 PR_EMS_AB_TOMBSTONE_LIFETIME
3
&H8146001 PR_EMS_AB_TRACKING_LOG_PATH_NAME
E
&H8147000 PR_EMS_AB_TRANS_RETRY_MINS
3

| | |
|-----------|--|
| &H8148000 | PR_EMS_AB_TRANS_TIMEOUT_MINS |
| 3 | |
| &H8149000 | PR_EMS_AB_TRANSFER_RETRY_INTERVAL |
| 3 | |
| &H814A000 | PR_EMS_AB_TRANSFER_TIMEOUT_NON_URGENT |
| 3 | |
| &H814B000 | PR_EMS_AB_TRANSFER_TIMEOUT_NORMAL |
| 3 | |
| &H814C000 | PR_EMS_AB_TRANSFER_TIMEOUT_URGENT |
| 3 | |
| &H814D000 | PR_EMS_AB_TRANSLATION_TABLE_USED |
| 3 | |
| &H814E000 | PR_EMS_AB_TRANSPORT_EXPEDITED_DATA |
| B | |
| &H814F000 | PR_EMS_AB_TRUST_LEVEL |
| 3 | |
| &H8150000 | PR_EMS_AB_TURN_REQUEST_THRESHOLD |
| 3 | |
| &H8151000 | PR_EMS_AB_TWO_WAY_ALTERNATE_FACILITY |
| B | |
| &H8152000 | PR_EMS_AB_UNAUTH_ORIG_BL_O |
| D | |
| &H8152101 | PR_EMS_AB_UNAUTH_ORIG_BL |
| E | |
| &H8153110 | PR_EMS_AB_USER_PASSWORD |
| 2 | |
| &H8154000 | PR_EMS_AB_USN_CREATED |
| 3 | |
| &H8155000 | PR_EMS_AB_USN_DSA_LAST_OBJ_REMOVED |
| 3 | |
| &H8156000 | PR_EMS_AB_USN_LAST_OBJ_REM |
| 3 | |
| &H8157000 | PR_EMS_AB_USN_SOURCE |
| 3 | |
| &H8158101 | PR_EMS_AB_X121_ADDRESS |
| E | |
| &H8159010 | PR_EMS_AB_X25_CALL_USER_DATA_INCOMING |
| 2 | |
| &H815A010 | PR_EMS_AB_X25_CALL_USER_DATA_OUTGOING |
| 2 | |
| &H815B010 | PR_EMS_AB_X25_FACILITIES_DATA_INCOMING |
| 2 | |
| &H815C010 | PR_EMS_AB_X25_FACILITIES_DATA_OUTGOING |
| 2 | |
| &H815D010 | PR_EMS_AB_X25_LEASED_LINE_PORT |
| 2 | |
| &H815E000 | PR_EMS_AB_X25_LEASED_OR_SWITCHED |
| B | |
| &H815F001 | PR_EMS_AB_X25_REMOTE_MTA_PHONE |

E
&H8160010 PR_EMS_AB_X400_ATTACHMENT_TYPE
2
&H8161000 PR_EMS_AB_X400_SELECTOR_SYNTAX
3
&H8162010 PR_EMS_AB_X500_ACCESS_CONTROL_LIST
2
&H8163000 PR_EMS_AB_XMIT_TIMEOUT_NON_URGENT
3
&H8164000 PR_EMS_AB_XMIT_TIMEOUT_NORMAL
3
&H8165000 PR_EMS_AB_XMIT_TIMEOUT_URGENT
3
&H8166010 PR_EMS_AB_SITE_FOLDER_GUID
2
&H8167000 PR_EMS_AB_SITE_FOLDER_SERVER_O
D
&H8167001 PR_EMS_AB_SITE_FOLDER_SERVER
E
&H8168000 PR_EMS_AB_REPLICATION_MAIL_MSG_SIZE
3
&H8169010 PR_EMS_AB_MAXIMUM_OBJECT_ID
2
&H8170101 PR_EMS_AB_NETWORK_ADDRESS
E
&H8171101 PR_EMS_AB_LDAP_DISPLAY_NAME
E
&H8173000 PR_EMS_AB_SCHEMA_FLAGS
3
&H8174000 PR_EMS_AB_BRIDGEHEAD_SERVERS_O
D
&H8174101 PR_EMS_AB_BRIDGEHEAD_SERVERS
E
&H8175001 PR_EMS_AB_WWW_HOME_PAGE
E
&H8176001 PR_EMS_AB_NNTP_CONTENT_FORMAT
E
&H8177001 PR_EMS_AB_POP_CONTENT_FORMAT
E
&H8178000 PR_EMS_AB_LANGUAGE
3
&H8179001 PR_EMS_AB_POP_CHARACTER_SET
E
&H817A000 PR_EMS_AB_USN_INTERSITE
3
&H817B001 PR_EMS_AB_SUB_SITE
E
&H817C100 PR_EMS_AB_SCHEMA_VERSION
3

&H817D001 PR_EMS_AB_NNTP_CHARACTER_SET
E
&H817E000 PR_EMS_AB_USE_SERVER_VALUES
B
&H817F000 PR_EMS_AB_ENABLED_PROTOCOLS
3
&H8180010 PR_EMS_AB_CONNECTION_LIST_FILTER
2
&H8181101 PR_EMS_AB_AVAILABLE_AUTHORIZATION_PACKAGES
E
&H8182101 PR_EMS_AB_CHARACTER_SET_LIST
E
&H8183000 PR_EMS_AB_USE_SITE_VALUES
B
&H8184101 PR_EMS_AB_ENABLED_AUTHORIZATION_PACKAGES
E
&H8185001 PR_EMS_AB_CHARACTER_SET
E
&H8186000 PR_EMS_AB_CONTENT_TYPE
3
&H8187000 PR_EMS_AB_ANONYMOUS_ACCESS
B
&H8188010 PR_EMS_AB_CONTROL_MSG_FOLDER_ID
2
&H8189001 PR_EMS_AB_USENET_SITE_NAME
E
&H818A010 PR_EMS_AB_CONTROL_MSG_RULES
2
&H818B001 PR_EMS_AB_AVAILABLE_DISTRIBUTIONS
E
&H818D010 PR_EMS_AB_OUTBOUND_HOST
2
&H818E101 PR_EMS_AB_INBOUND_HOST
E
&H818F000 PR_EMS_AB_OUTGOING_MSG_SIZE_LIMIT
3
&H8190000 PR_EMS_AB_INCOMING_MSG_SIZE_LIMIT
3
&H8191000 PR_EMS_AB_SEND_TNEF
B
&H8192010 PR_EMS_AB_AUTHORIZED_PASSWORD_CONFIRM
2
&H8193001 PR_EMS_AB_INBOUND_NEWSFEED
E
&H8194000 PR_EMS_AB_NEWSFEED_TYPE
3
&H8195001 PR_EMS_AB_OUTBOUND_NEWSFEED
E
&H8196010 PR_EMS_AB_NEWSGROUP_LIST

2
&H8197101 PR_EMS_AB_NNTP_DISTRIBUTIONS
E
&H8198001 PR_EMS_AB_NEWSGROUP
E
&H8199001 PR_EMS_AB_MODERATOR
E
&H819A001 PR_EMS_AB_AUTHENTICATION_TO_USE
E
&H819B000 PR_EMS_AB_HTTP_PUB_GAL
B
&H819C000 PR_EMS_AB_HTTP_PUB_GAL_LIMIT
3
&H819E110 PR_EMS_AB_HTTP_PUB_PF
2
&H81A1001 PR_EMS_AB_X500_RDN
E
&H81A2001 PR_EMS_AB_X500_NC
E
&H81A3101 PR_EMS_AB_REFERRAL_LIST
E
&H81A4000 PR_EMS_AB_NNTP_DISTRIBUTIONS_FLAG
B
&H81A5000 PR_EMS_AB_ASSOC_PROTOCOL_CFG_NNTP_O
D
&H81A5001 PR_EMS_AB_ASSOC_PROTOCOL_CFG_NNTP
E
&H81A6000 PR_EMS_AB_NNTP_NEWSFEEDS_O
D
&H81A6101 PR_EMS_AB_NNTP_NEWSFEEDS
E
&H81A8000 PR_EMS_AB_ENABLED_PROTOCOL_CFG
B
&H81A9101 PR_EMS_AB_HTTP_PUB_AB_ATTRIBUTES
E
&H81AB101 PR_EMS_AB_HTTP_SERVERS
E
&H81AC000 PR_EMS_AB_MODERATED
B
&H81AD001 PR_EMS_AB_RAS_ACCOUNT
E
&H81AE010 PR_EMS_AB_RAS_PASSWORD
2
&H81AF010 PR_EMS_AB_INCOMING_PASSWORD
2
&H81B0000 PR_EMS_AB_OUTBOUND_HOST_TYPE
B
&H81B1000 PR_EMS_AB_PROXY_GENERATION_ENABLED
B

&H81B2010 PR_EMS_AB_ROOT_NEWSGROUPS_FOLDER_ID
2
&H81B3000 PR_EMS_AB_CONNECTION_TYPE
B
&H81B4000 PR_EMS_AB_CONNECTION_LIST_FILTER_TYPE
3
&H81B5000 PR_EMS_AB_PORT_NUMBER
3
&H81B6101 PR_EMS_AB_PROTOCOL_SETTINGS
E
&H81B7001 PR_EMS_AB_GROUP_BY_ATTR_1
E
&H81B8001 PR_EMS_AB_GROUP_BY_ATTR_2
E
&H81B9001 PR_EMS_AB_GROUP_BY_ATTR_3
E
&H81BA001 PR_EMS_AB_GROUP_BY_ATTR_4
E
&H81BE001 PR_EMS_AB_VIEW_SITE
E
&H81BF001 PR_EMS_AB_VIEW_CONTAINER_1
E
&H81C0001 PR_EMS_AB_VIEW_CONTAINER_2
E
&H81C1001 PR_EMS_AB_VIEW_CONTAINER_3
E
&H81C2004 PR_EMS_AB_PROMO_EXPIRATION
0
&H81C3101 PR_EMS_AB_DISABLED_GATEWAY_PROXY
E
&H81C4010 PR_EMS_AB_COMPROMISED_KEY_LIST
2
&H81C5000 PR_EMS_AB_INSADMIN_O
D
&H81C5001 PR_EMS_AB_INSADMIN
E
&H81C6000 PR_EMS_AB_OVERRIDE_NNTP_CONTENT_FORMAT
B
&H81C7000 PR_EMS_AB_OBJ_VIEW_CONTAINERS_O
D
&H81C7101 PR_EMS_AB_OBJ_VIEW_CONTAINERS
E
&H8C18000 PR_EMS_AB_VIEW_FLAGS
3
&H8C19001 PR_EMS_AB_GROUP_BY_ATTR_VALUE_STR
E
&H8C1A000 PR_EMS_AB_GROUP_BY_ATTR_VALUE_DN_O
D
&H8C1A001 PR_EMS_AB_GROUP_BY_ATTR_VALUE_DN

E
&H8C1B110 PR_EMS_AB_VIEW_DEFINITION
2
&H8C1C010 PR_EMS_AB_MIME_TYPES
2
&H8C1D000 PR_EMS_AB_LDAP_SEARCH_CFG
3
&H8C1E000 PR_EMS_AB_INBOUND_DN_O
D
&H8C1E001 PR_EMS_AB_INBOUND_DN
E
&H8C1F000 PR_EMS_AB_INBOUND_NEWSFEED_TYPE
B
&H8C20000 PR_EMS_AB_INBOUND_ACCEPT_ALL
B
&H8C21000 PR_EMS_AB_ENABLED
B
&H8C22000 PR_EMS_AB_PRESERVE_INTERNET_CONTENT
B
&H8C23000 PR_EMS_AB_DISABLE_DEFERRED_COMMIT
B
&H8C24000 PR_EMS_AB_CLIENT_ACCESS_ENABLED
B
&H8C25000 PR_EMS_AB_REQUIRE_SSL
B
&H8C26001 PR_EMS_AB_ANONYMOUS_ACCOUNT
E
&H8C27010 PR_EMS_AB_CERTIFICATE_CHAIN_V3
2
&H8C28010 PR_EMS_AB_CERTIFICATE_REVOCATION_LIST_V3
2
&H8C29010 PR_EMS_AB_CERTIFICATE_REVOCATION_LIST_V1
2
&H8C30110 PR_EMS_AB_CROSS_CERTIFICATE_CRL
2
&H8C31000 PR_EMS_AB_SEND_EMAIL_MESSAGE
B
&H8C32000 PR_EMS_AB_ENABLE_COMPATIBILITY
B
&H8C33101 PR_EMS_AB_SMIME_ALG_LIST_NA
E
&H8C34101 PR_EMS_AB_SMIME_ALG_LIST_OTHER
E
&H8C35001 PR_EMS_AB_SMIME_ALG_SELECTED_NA
E
&H8C36001 PR_EMS_AB_SMIME_ALG_SELECTED_OTHER
E
&H8C37000 PR_EMS_AB_DEFAULT_MESSAGE_FORMAT
B

&H8C38001 PR_EMS_AB_TYPE
E
&H8C3A000 PR_EMS_AB_DO_OAB_VERSION
3
&H8C3B010 PR_EMS_AB_VOICE_MAIL_SYSTEM_GUID
2
&H8C3C001 PR_EMS_AB_VOICE_MAIL_USER_ID
E
&H8C3D001 PR_EMS_AB_VOICE_MAIL_PASSWORD
E
&H8C3E010 PR_EMS_AB_VOICE_MAIL_RECORDED_NAME
2
&H8C3F101 PR_EMS_AB_VOICE_MAIL_GREETINGS
E
&H8C40110 PR_EMS_AB_VOICE_MAIL_FLAGS
2
&H8C41000 PR_EMS_AB_VOICE_MAIL_VOLUME
3
&H8C42000 PR_EMS_AB_VOICE_MAIL_SPEED
3
&H8C43100 PR_EMS_AB_VOICE_MAIL_RECORDING_LENGTH
3
&H8C44001 PR_EMS_AB_DISPLAY_NAME_SUFFIX
E
&H8C45110 PR_EMS_AB_ATTRIBUTE_CERTIFICATE
2
&H8C46110 PR_EMS_AB_DELTA_REVOCATION_LIST
2
&H8C47110 PR_EMS_AB_SECURITY_POLICY
2
&H8C48000 PR_EMS_AB_SUPPORT_SMIME_SIGNATURES
B
&H8C49000 PR_EMS_AB_DELEGATE_USER
B
&H8C50000 PR_EMS_AB_LIST_PUBLIC_FOLDERS
B
&H8C51001 PR_EMS_AB_LABELEDURI
E
&H8C52000 PR_EMS_AB_RETURN_EXACT_MSG_SIZE
B
&H8C53001 PR_EMS_AB_GENERATION_QUALIFIER
E
&H8C54001 PR_EMS_AB_HOUSE_IDENTIFIER
E
&H8C55010 PR_EMS_AB_SUPPORTED_ALGORITHMS
2
&H8C56001 PR_EMS_AB_DMD_NAME
E
&H8C57001 PR_EMS_AB_EXTENSION_ATTRIBUTE_11

E
&H8C58001 PR_EMS_AB_EXTENSION_ATTRIBUTE_12
E
&H8C59001 PR_EMS_AB_EXTENSION_ATTRIBUTE_13
E
&H8C60001 PR_EMS_AB_EXTENSION_ATTRIBUTE_14
E
&H8C61001 PR_EMS_AB_EXTENSION_ATTRIBUTE_15
E
&H8C62000 PR_EMS_AB_REPLICATED_OBJECT_VERSION
3
&H8C63001 PR_EMS_AB_MAIL_DROP
E
&H8C64001 PR_EMS_AB_FORWARDING_ADDRESS
E
&H8C65010 PR_EMS_AB_FORM_DATA
2
&H8C66001 PR_EMS_AB_OWA_SERVER
E
&H8C67001 PR_EMS_AB_EMPLOYEE_NUMBER
E
&H8C68001 PR_EMS_AB_TELEPHONE_PERSONAL_PAGER
E
&H8C69001 PR_EMS_AB_EMPLOYEE_TYPE
E
&H8C6A110 PR_EMS_AB_TAGGED_X509_CERT
2
&H8C6B001 PR_EMS_AB_PERSONAL_TITLE
E
&H8C6C001 PR_EMS_AB_LANGUAGE_ISO639
E
&HF000000 PR_EMS_AB_OTHER_RECIPS
D
&HFFF8101 PR_EMS_AB_CHILD_RDNS
E
&HFFF9001 PR_EMS_AB_HIERARCHY_PATH
E
&HFFFA010 PR_EMS_AB_OBJECT_OID
2
&HFFFB000 PR_EMS_AB_IS_MASTER
B
&HFFFC010 PR_EMS_AB_PARENT_ENTRYID
2
&HFFFD000 PR_EMS_AB_CONTAINERID
3
&HFFFD000 PR_EMS_AB_DOS_ENTRYID
3
&HFFFE001 PR_EMS_AB_SERVER
E

Web Page Support

The CDO libraries offer programmatic support for Hypertext Markup Language (HTML) script on a Web page. Script can be server-side, which is decoded and run at the Web server, or client-side, which is decoded and run at the browser. The CDO Rendering Library supports server-side script, and the CDO Library supports client-side script. For more information on server-side script support, see [HTML Rendering](#).

The language used for the script subroutine can be Microsoft® Visual Basic® Scripting Edition (VBScript), Microsoft® JScript™, or JavaScript. For simplicity of browser implementation, and also for security reasons, not all Visual Basic functionality is available in VBScript. In particular, you cannot use:

- File input/output or direct operating system access.
- Early binding, for example `Dim objRecip As Recipient`.
- Type library constants, such as **CdoPR_DISPLAY_TYPE**.
- Named parameters in calls to methods.
- Dialog boxes, for example in the [Logon](#) method.

For more information on VBScript and its feature restrictions, see the *Microsoft Visual Basic Scripting Edition Language Reference*.

To use the CDO Library in client-side script, you instantiate a CDO [Session](#) object in the body of a Web page, and then log on to this session from a script subroutine. Following logon, you can instantiate and use other objects subsidiary to the session object.

During logon, a MAPI client can only reference a profile that is stored in its own local MAPISVC.INF configuration file. A browser is not likely to be able to access a profile defined and stored at the Web server. Therefore, script running at a browser should create a profile dynamically so that it is local to the browser. This can be done by using the *ProfileInfo* parameter of the [Logon](#) method, or, as in the following example, by calling **Logon** without parameters and letting the browser user choose a profile.

Dynamic profile creation is only possible on a message service that is tightly coupled with MAPI, such as the Microsoft® Exchange server. Loosely coupled message services cause the MAPI spooler to be started, and if another message service or user tries to access MAPI, serious errors result.

This client-side script fragment sends a feedback message to the Customer Support department when the user clicks the **Send Feedback** button. It demonstrates a script subroutine in VBScript showing instantiation of a Session object with an HTML <OBJECT> tag. It also shows the subsequent instantiation of a [Message](#) object and its [Recipients](#) collection, and the preparation and submission of an e-mail message.

```
<HTML>
<BODY LANGUAGE=VBS>
...
<SCRIPT LANGUAGE=VBS>
Sub Send_Feedback
    Dim objFBMess ' feedback message from Web page
    Dim objRecips ' can't do "Dim As" (early binding) in VBScript
' ... validate objWebSession object instantiated by HTML, then ...
    objWebSession.Logon ' let user choose profile if not logged on
    Set objFBMess = objWebSession.Outbox.Messages.Add
    objFBMess.Subject = "Feedback from Web page"
    objFBMess.Text = Feedback.Value
    Set objRecips = objFBMess.Recipients
    objRecips.Add "custsupp" ' send to Customer Support
    objRecips.Resolve
```

```
    objFBMess.Send ' defaults to save copy and no user dialog
    objWebSession.Logoff
End Sub
</SCRIPT>
...
<H1><CENTER>CUSTOMER FEEDBACK WEB PAGE</H1>
<B><P>Welcome to the Customer Support Feedback Web page.
<P>If you have any additional suggestions or requests,
<P>please send
<A HREF=MAILTO:custsupp@mycompany.com>Customer Support</A>
some e-mail.
<P>Please enter your feedback here:
<INPUT NAME=Feedback TYPE=Text SIZE=80>
<INPUT ONCLICK=Send_Feedback TYPE=Button VALUE="Send Feedback">
<OBJECT CLASSID="clsid:3FA7DEB3-6438-101B-ACC1-00AA00423326"
ID=objWebSession>
<! The OBJECT tag instantiates the CDO Session object>
</OBJECT>
</BODY>
</HTML>
```

Note The <OBJECT> and <SCRIPT> tags and the <LANGUAGE> and <ONCLICK> attributes are defined in the HTML 3.2 specification of the World Wide Web Consortium (W3C). Not all Internet browsers support HTML 3.2 or all of its elements. For more information on HTML 3.2, see the Web page *HTML 3.2 Reference Specification* at <http://www.w3.org/pub/WWW/TR/REC-html32.html>.

For more information on the programming elements in the script fragment, see the Session object's Logon and Logoff methods, the Messages collection's Add method, the Message object's Recipients property and Send method, and the Recipients collection's Add and Resolve methods.

How Programmable Objects Work

How do programmable objects work? How do the CDO libraries offer their powerful ability to create, manage, and render messaging objects?

This appendix provides a very short introduction to the Microsoft® Component Object Model (COM), Automation, and the OLE programmability interface **IDispatch**. For complete details, see the "COM and ActiveX Object Services" section of the Microsoft Platform SDK.

You do not need to understand this material in order to use the CDO libraries.

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 functions 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 Microsoft® 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 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**, **AddRef**, and **Release**, in that order. 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 IDL 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 Microsoft® 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 COM model, an object obtains a pointer to an interface and then calls a member method of the interface. With **IDispatch**, instead of directly calling the member method, 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 data type, known also as *variant type*, contains a constant such as VT_I2 or VT_DATE, which defines valid values for the data types. For more information on variant types, see the **Type** property of the **Field** object.

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.

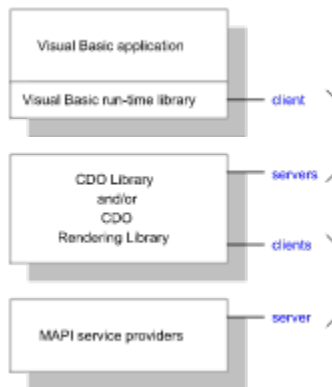
An executable file or DLL that exposes **IDispatch** and its type library is known as an *Automation server*. The CDO Library and the CDO Rendering Library are both Automation servers. In version 1.1 and later they are also in-process servers, residing in .DLL files and linking dynamically with the calling modules.

The CDO Libraries as Automation Servers

So, let's put it all together, from the bottom up, to see how the CDO libraries work.

- Service providers implement COM interfaces – specifically, the MAPI interfaces – as described in the MAPI documentation.
- The CDO libraries implement several objects (Session, Message, ContainerRenderer, Format, and so on) that act as *clients* to these MAPI interfaces. That is, the CDO library objects obtain pointers to the MAPI interfaces and call MAPI methods.
- The CDO libraries implement **IDispatch** and act as Automation servers so that they can be called by tools that can use **IDispatch**, such as Microsoft® Visual Basic®. That is, they allow other programs to call them through the **IDispatch** interface. Beginning with version 1.1, the CDO libraries are self-registering as Automation servers.
- The CDO libraries publish type libraries that contain information about the objects they make available through **IDispatch**.
- Your Visual Basic application acts as a client to the CDO libraries. It reads their type libraries to obtain information about their objects, methods, and properties. When your Visual Basic application declares a variable as an object (with code such as `Dim objSession as Object` or `Dim objSession as MAPI.Session`) 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 one or both of the CDO libraries, which are the Automation servers. The CDO libraries, in turn, act as clients to the MAPI services.



The CDO Libraries and MAPI

The CDO libraries call Microsoft COM and MAPI interfaces for you. The following table describes the MAPI interfaces that the CDO Library calls when you manipulate a CDO Library object.

| CDO Library object | COM or MAPI interface called by the CDO Library |
|---------------------------|---|
| AddressEntry | IMailUser |
| AddressEntryFilter | IMAPITable (with Restrict method) |
| AddressList | IABContainer |
| Attachment | IAttach |
| Field | IMAPIProp, IStream |
| Folder | IMAPIFolder |
| InfoStore | IMsgStore |
| Message | IMessage |
| MessageFilter | IMAPITable (with Restrict method) |
| Recipient | IMAPITable row (recipient table from IMessage) |
| Session | IMAPISession |

For collection objects, the CDO libraries call the MAPI interface **IMAPITable**.

The CDO libraries also call the MAPI interface **IMAPIProp**. Many of the properties exposed by the CDO libraries are based on MAPI properties. The following table describes the mapping between these CDO library properties and the underlying MAPI properties.

| CDO 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 | ProviderName | PR_PROVIDER_DISPLAY | 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 | Address | combination of PR_ADDRTYPE and PR_EMAIL_ADDRESS | PT_TSTRING |
| 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:

- "COM and ActiveX Object Services" section of the Microsoft Platform SDK
- *Inside OLE, Second Edition*, by Kraig Brockschmidt, published by Microsoft Press

Note that this document contains the latest known information about the Microsoft® CDO libraries at the time of publication. Where terms in this document differ from other Microsoft® Visual Basic®, OLE, or Component Object Model (COM) terms, this document should be viewed as the definition of the specific implementation represented by the CDO libraries.

Java Programming Considerations

Accessing the CDO Library and CDO Rendering Library objects from Java requires different procedures from those used with Microsoft® Visual Basic®. This section describes the important differences and provides some examples of Java access.

You do not need to understand this material if you are only using Visual Basic for the CDO libraries.

CDO Classes in Java

To take advantage of the CDO libraries in a Java program, you must first create Java class wrappers from their type libraries. The type library for the CDO Library is embedded in CDO.DLL. The type library for the CDO Rendering Library is embedded in CDOHTML.DLL.

Group

To create a class wrapper

1. Run the Microsoft® Visual J++™ Development Environment.
2. From the **Tools** menu, choose the **Java Type Library Wizard**.
3. From the list of available Automation servers, select **Microsoft CDO 1.2 Library** or **Microsoft CDO 1.2 Rendering Library**.
4. Click **OK** to tell the wizard to create the needed classes.

The preceding steps cause the Java classes to reside in the subdirectory ...\\Java\\TrustLib\\cdo or ...\\Java\\TrustLib\\cdohtml, depending on your choice of Automation server. You may wish to print out the SUMMARY.TXT file located in this directory. It is a useful reference for Java support for the CDO libraries.

Java Language Features

The Java language is based on C++ and shows considerable similarity to it. In particular, Java code is case-sensitive. There are, however, significant differences between the two languages.

Because it is interpreted instead of compiled, Java cannot process **#define** directives. This means that the constants defined in the CDO and CDO Rendering type libraries cannot be used in a Java program. You have to use numeric equivalents, which can be found in the [Error Codes](#), [MAPI Property Tags](#), and [Microsoft Exchange Property Tags](#) appendices.

Java does not provide any error trapping mechanism equivalent to the Microsoft® Visual Basic® **On Error GoTo** statement. All errors must be anticipated and explicitly tested for after each call that could generate them.

Like all alphanumeric elements in Java, the keywords are case-sensitive. Their predefined values are all lowercase, such as **true**, **false**, and **null**. This means Java does not recognize capitalized keywords such as **True**, **False**, or **Null**, which you may be accustomed to using in Visual Basic.

Java objects expose only methods and no properties. CDO library properties are referenced through accessor methods defined for each property by prefixing **get** or **put** to the property name, for example **getInbox** and **putSubject**. Parameters to methods are referenced by accessor methods with **get** or **put** prefixed to the data type, such as **getString** and **putBoolean**.

Read-only properties use the corresponding **get** accessor method. The **Inbox** property of the **Session** object, for example, can be read with the **getInbox** method:

```
Variant inboxFolder = new Variant();
inboxFolder = session.getInbox();
```

Read/write properties use the corresponding **get** and **put** accessor methods. For example, the **Subject** property of the **Message** object can be read with **getSubject** and written with **putSubject**, after the parameter is prepared with the help of the **getString** and **putString** methods:

```
Variant inSubject = new Variant();
StringBuffer newSubject = new StringBuffer( "RE: " );
Variant outSubject = new Variant();
...
inSubject = inMessage.getSubject();
newSubject.append( inSubject.getString() ); // already have "RE: "
outSubject.putString( newSubject.toString() );
outMessage.putSubject( outSubject );
```

The property accessor methods such as **getInbox** always return a Variant object. If your code assigns them to another Variant object, as in the preceding code fragments, the types already match. If, however, you cast an object returned from an accessor method to another object type, you need to use the **getDispatch** method to obtain a type match:

```
Folder inboxFolder;
inboxFolder = (Folder) session.getInbox().getDispatch();
```

The CDO libraries support the **IDispatch** interface, which allows a program to access the underlying messaging and rendering objects. Invocation of the **getDispatch** method signals Java to call **QueryInterface** on a Variant object and obtain the appropriate messaging or rendering interface for it. For more information, see [IDispatch](#).

Java methods do not allow for optional parameters. Calls to methods must present every parameter included in the definition of the method. The equivalent of a Visual Basic **Null** parameter can be achieved by using the **noParam** method on a Java Variant object:


```
Variant nullPar = new Variant();  
nullPar.noParam();  
object.Method( firstPar, nullPar, thirdPar, nullPar, nullPar );
```

Java Programming Examples

This section presents code illustrating Java access to the CDO Library and the CDO Rendering Library. Each example accomplishes a task your program might perform when dealing with CDO library objects.

The following table summarizes the programming procedures used in the examples. Note that all tasks require creation of a valid session and a successful logon.

| Programming task | Procedure |
|---------------------------------------|--|
| <u>Logging On and Off</u> | <ol style="list-style-type: none">1. Create a session.2. Log on.3. Log off. |
| <u>Creating and Sending a Message</u> | <ol style="list-style-type: none">1. Create a session.2. Log on.3. Get the Outbox Messages collection.4. Add a new message.5. Populate the new message.6. Send the new message.7. Log off. |

It is important to understand the object hierarchies of the CDO libraries, because they determine the correct syntax of Java statements. The relative positioning of the objects within the hierarchies determines the order in which the objects should be accessed. For more information on the hierarchies, see [Object Model](#) and [Rendering Object Model](#).

To access the CDO Library, a Java program should import CDO.DLL on 32-bit platforms. To access the CDO Rendering Library, import CDOHTML.DLL on 32-bit platforms. Neither library is available on 16-bit platforms.

Logging On and Off (Java) Group

This example is a standalone application, not an applet. It is a complete program, including the importation declarations and the **main()** function. It demonstrates the basic framework of a Java program that accesses CDO Library objects. The **main()** function creates a Session object, logs on to it, and logs off, with minimal error checking.

```
import cdo.*;
import com.ms.com.*;

public class Sample0
{
    public static void main(String args[])
    {
        // Create a messaging session (call the class factory).
        _Session session = (_Session) new Session();

        Variant v = new Variant();
        Variant profileName = new Variant();
        profileName.putString( "MyProfile" );
        v.noParam();

        // Log on.
        try
        {
            session.Logon( profileName, v, v, v, v, v, v );
        }
        catch( Exception e )
        {
            System.out.println( "Logon failed!" );
        }

        // The logon was successful; now do desired processing.

        // Now we will log off.
        try
        {
            session.Logoff();
        }
        catch( Exception e )
        {
            System.out.println( "Logoff failed!" );
        }
    }
}
```

Creating and Sending a Message (Java)

Group

This example is a standalone application, not an applet. It is a complete program demonstrating the basic procedures for initiating a CDO Library message. Within the framework of logging on to and off from a session, it creates a Message object, sets its requisite properties, and sends it to its recipients through the MAPI system.

```
import cdo.*;
import com.ms.com.*;
import java.io.*;

public class Sample1
{
    public static void main(String args[])
    {
        // Call the class factory.
        _Session session = (_Session) new Session();

        Variant nullParam = new Variant();
        nullParam.noParam();
        Messages messages = null;

        try    // First, log on.
        {
            Variant profileName = new Variant();
            profileName.putString( "MyProfile" );
            session.Logon( profileName, nullParam, nullParam, nullParam,
                           nullParam, nullParam, nullParam );
        }
        catch( Exception e )
        {
            System.out.println( "Logon failed!" );
            e.printStackTrace();
        }

        try    // Get the messages collection in the outbox.
        {
            Folder outBox = (Folder) session.getOutbox().getDispatch();
            messages = (Messages) outBox.getMessages().getDispatch();
        }
        catch( Exception e )
        {
            System.out.println( "Failed to get the outbox messages collection!" );
            e.printStackTrace();
        }

        try    // Now create, populate, and send the message.
        {
            // Create a new message in the outbox.
            Message newMsg = (Message) messages.Add( nullParam, nullParam,
                                                       nullParam, nullParam ).getDispatch();

            // Set the subject of the new message.
            Variant subject = new Variant();
```

```

subject.putString( "This is a message from Java!" );
newMsg.putSubject( subject );

// Set the text (body) of the new message.
Variant text = new Variant();
text.putString( "This is the message text." );
newMsg.putText( text );

// Get the Recipients collection.
Recipients recip = (Recipients) newMsg.getRecipients().getDispatch();

// Create a new recipient in the Recipients collection.
Recipient recip = (Recipient) recip.Add( nullParam, nullParam,
                                         nullParam, nullParam ).getDispatch();

// Set the name on the new recipient.
Variant name = new Variant();
name.putString( "somebody" );
recip.putName( name );

// Set the type on the new recipient.
Variant type = new Variant();
type.putInt( 1 ); // "To" recipient
recip.putType( type );

// Resolve the new recipient.
Variant showDialog = new Variant();
showDialog.putBoolean( true );
recip.Resolve( showDialog );

// Make the new message permanent.
Variant refresh = new Variant();
Variant makePermanent = new Variant();
refresh.putBoolean( true );
makePermanent.putBoolean( true );
newMsg.Update( makePermanent, refresh );

// Send the new message.
Variant saveCopy = new Variant();
saveCopy.noParam();
showDialog.noParam(); // already created under Resolve()
Variant wndHandle = new Variant();
wndHandle.noParam();
newMsg.Send( saveCopy, showDialog, wndHandle );
}
catch( Exception e )
{
    System.out.println( "Failed to create/send a new message!" );
}

try    // Finally, log off.
{
    session.Logoff();
}
catch( Exception e )

```

```
{
  System.out.println( "Logoff failed!" );
}
System.out.println( "All done!" );
}
}
```

Common Mistakes

Although the CDO libraries are remarkably free of programming pitfalls, there are some aspects of Microsoft® Visual Basic® syntax that can lead to misunderstanding. This appendix explains some common design and coding errors.

Reinstantiating an Object

You normally instantiate a CDO library object by accessing a property on a parent object that causes the desired object to be created. For example, the statement

```
Set objMsg = objSession.Inbox.Messages.Item(1)
```

instantiates a Folder object for the Inbox property of the Session object, then a Messages collection object for the Messages property of the Inbox folder, and then a Message object for the Item property of the Messages collection.

Every time you cross a period from left to right, you instantiate the object on the right of the period. This is true whether or not you have previously instantiated another version of the same object. This code fragment, for example, is intended to count the members of an AddressEntries collection that can resolve the name "John":

```
Dim objAdrList As AddressList
Dim objAEFilt As AddressEntryFilter
Dim objAE As AddressEntry
...
Set objAEFilt = objAdrList.AddressEntries.Filter
objAEFilt.Name = "John" ' set filter to restrict on this name
i = 0
For Each objAE in objAdrList.AddressEntries
    i = i + 1
    MsgBox objAE.Name
Next
```

As written, however, this code counts every AddressEntry object in the collection. This is because the collection itself is instantiated twice, once when setting the filter and once when initializing the loop, in response to the code `objAdrList.AddressEntries`. The second collection is instantiated with a default AddressEntryFilter object with no restrictions, which is used in the loop. The filter with the Name property restriction remains with the first collection and is never used.

This behavior is counterintuitive to programmers accustomed to having the same object returned by repeated references. But no variable is defined for an object that is generated internally by crossing a period, and Visual Basic has no way of correlating its internal objects. It is up to you to take care of the correlation at the source code level.

The proper approach is to define and **Set** a variable for any object you plan to use more than once. In the case of the previous code fragment, it is the AddressEntries collection object that is to be reused:

```
Dim colAddrEntries As AddressEntries
...
Set colAddrEntries = objAdrList.AddressEntries
Set objAEFilt = colAddrEntries.Filter
objAEFilt.Name = "John" ' set filter to restrict on this name
i = 0
For Each objAE in colAddrEntries
    i = i + 1
    MsgBox objAE.Name
Next
```

C/C++ programmers should observe that the period in Visual Basic is not a class member access operator, nor is it a pointer. Although `objAdrList.AddressEntries` may appear analogous to a construction such as `lpAdrList->AddressEntries`, there is in fact very little parallelism between the two, and access to a

property in C/C++ requires more than a single statement.

When you instantiate an object multiple times, you subject your application to several problems:

- Your application wastes execution time and memory creating and retaining more than one version of the object.
- A subsequent instantiation does not **Release** a previous one, and all instantiated objects remain in memory with nonzero reference counts.
- The instantiations are unrelated to one another, and any operations you perform on one have no effect on any of the others.
- A given instantiation is used only by the code that follows its creation and precedes a subsequent instantiation. That is, different sections of your code are using different, unrelated versions of the object.

The safest procedure is to use explicit variables for all the objects and collections in your application. The consequence of not doing so can vary from inefficient execution to wrong results. For more information, see [Improving Application Performance](#).

Looping Through a Collection

A CDO collection is always refreshed immediately following an **Add** or **Delete** operation on any of its members. This means that the collection's **Count** property is incremented or decremented, and all the members following the point of insertion or deletion are reindexed. To access one of these members, you must use its new index value. This is easy to forget if you are looping through the collection.

Consider this code fragment, intended to delete every member of a Messages collection:

```
Dim colMsgs As Messages
...
size = colMsgs.Count
For i = 1 to size
    colMsgs.Item(i).Delete
Next i
```

When $i = 1$, the first Message object's **Delete** method is called and that message is deleted. The Messages collection's **Count** property is immediately decremented by one, and the Message object that had been the second message in the collection now becomes the first. Therefore, when $i = 2$, the second message in the reindexed collection, that is, the message that was originally third in the collection, is deleted, and the message that was originally fourth now becomes second.

The effect of this loop is to delete all the odd-numbered members of the original collection. Once i has been incremented past half the value of $size$, it becomes too large for the reindexed items, and a value of **Nothing** is returned for the remaining accesses.

The Microsoft® Visual Basic® **For Each** statement cannot be used as a workaround, because it is internally implemented as

```
For i = 1 to .Count
    .Item(i).
Next i
```

which still exhibits the skipping behavior, although it does at least reread the **Count** property with each passage of the loop, so that it avoids going past the end of the collection.

A loop based on **GetFirst** and **GetNext** also encounters the same problem. The collection is reindexed following a **Delete** call, but the internal pointer is not decremented, so it now points to the member after the deleted member. When **GetNext** is called, it increments the pointer and returns the member following the desired one.

A safer approach is to delete the first message in every iteration. This loop has the effect intended by the erroneous code fragment:

```
size = colMsgs.Count
For i = 1 to size
    colMsgs.Item(1).Delete
Next i
```

This loop also processes the messages in forward order, that is, starting with the first and ending with the last. Of course, if you wish to delete everything in the collection, the simplest alternative is to use the **Delete** method of the Messages collection itself.

If you need to examine all the members of a collection and selectively delete certain ones, you can iterate backward to avoid the problem of skipping over the member following a deleted member. This code fragment loops backward without using **Step -1**, because **Step** is not available in VBScript:

```
size = colMsgs.Count
```

```
For i = 1 to size
  Set objMsg = colMsgs.Item(size - i + 1)
  If objMsg.TimeExpired <= Now Then
    objMsg.Delete
  End If
Next i
```

Some collections, such as Columns, allow a new member to be inserted following a specified existing member instead of being added at the end. If you insert a Column object into the middle of the collection using its **Add** method, you must take into account the new **Index** values of the columns that come after the new column.

However, the **Add** method of any sortable collection can effectively insert a new member before the end. An AddressEntries collection, for example, is normally sorted on the **Name** property. If you call **Add** with a display name lying somewhere within the alphabet, the AddressEntry objects that follow it alphabetically are all reindexed.

Property Parameters

The CDO libraries interface with the Microsoft® Visual Basic® library and with the **IDispatch** interface. These interfaces define only methods and no properties. In order to expose properties on its objects, CDO implements all properties with accessor methods. A read/write property has corresponding **get** and **put** methods defined for it. A read-only property has a **get** method but no **put** method. For the most part, these accessor methods are transparent to a Visual Basic programmer.

The **get** and **put** methods are all defined with optional parameters. These are used by a Visual Basic programmer with properties that have parameterized syntax, such as the **Fields** property of the Attachment object or the **Item** property of the Views collection. In most properties other than **Fields** and **Item**, the syntax does not make use of any parameters. They can be supplied but will be ignored:

```
stFoldID = obFolder.ID(3, size, "All these parameters are ignored")
```

Because of the possibility of parameters, parentheses immediately following a property name are always construed to enclose a parameter list.

Array properties pose a special problem in this connection. A programmer intuitively expects to select a particular element of an array by supplying an index immediately after the array name. This works normally when the variable has been dimensioned as an array:

```
Dim arValues (100) As Integer ' array of 100 integers
' ...
sum = 0
For i = 1 To 100
    sum = sum + arValues(i)
Next i
```

But when the variable is a CDO property, the first set of parentheses after its name is taken as holding one or more syntactic parameters. If the property has an array data type, such as the **Categories** property of the Message object, a second set of parentheses must be added to specify the array index:

```
stPrincipalCategory = obMessage.Categories()(1)
```

This applies both when reading an array property and when writing it:

```
obMessage.Categories()(index) = ""
```

However, no parentheses are needed if the entire array is participating in an operation, because no index needs to be specified and there is no need for any parentheses at all:

```
Dim stCategories (10) As String
obMessage.Categories = stCategories ' full array copy
```

Glossary

A

Active Messaging

See [CDO](#).

Active Platform

A PC platform for developing [Internet](#) applications, including an extensive set of development tools. The platform is independent of the operating system and presents a consistent interface to both the client and the server. The Active Platform™ is based on three core technologies: Active Desktop, [Active Server™](#), and [ActiveX®](#).

Active Server

The [Active Platform™](#) component of Microsoft Internet Information Server (IIS) 3.0 or later, which extends the Windows NT system services to the [World Wide Web](#). It decodes and runs [server-side script](#) and provides database access and transaction support.

Active Server Pages

See [ASP](#).

ActiveX

A software technology, built on the [COM](#) foundation, that allows networked components to interact with each other independently of the languages they are written in. ActiveX® is used particularly for applications dealing with the [Internet](#) and the [World Wide Web](#). ActiveX components include [ActiveX objects](#), which expose their properties and methods, and [ActiveX clients](#), which access them. See *also* [OLE](#).

ActiveX client

An application or programming tool accessing the [ActiveX objects](#) exposed by programs supporting [Automation](#).

ActiveX control

A reusable, stand-alone software component often exposing a discrete subset of the total functionality of a product or application. An arbitrary number of ActiveX® controls can be used as prefabricated components to aid in building a new application. ActiveX controls cannot run alone and must be loaded into a control container such as Microsoft Visual Basic or the Microsoft Internet Explorer. Formerly referred to as OLE control or OCX. For more information, see the ActiveX SDK section of the Platform SDK.

ActiveX object

An [object](#) exposed by an application or programming tool supporting [Automation](#) for use by [ActiveX clients](#).

address book

A [container object](#) that manages a collection of one or more [address book containers](#) furnished by one or more service providers. CDO applications can access an address book using the [AddressLists](#) collection object.

address book container

An [object](#) that contains one or more [recipients](#) and makes them available to applications using the [CDO libraries](#). Common address book containers include the [global address list](#) and the [personal address book](#). CDO applications can access an address book container using the [AddressList](#) object.

address entry

An object containing addressing information such as a display name, an e-mail type, and an e-mail address. An address usually represents a person or process that can receive a message. CDO applications can access an address entry using the AddressEntry object. See *also* recipient.

anonymous user

See unauthenticated user.

ASP

(Active Server Pages) An open application environment in which HTML pages, scripts, and ActiveX components can be combined to create Web-based applications. ASP is an ISAPI application.

attachment

An object that is associated with a message and contains additional data, such as a file or an OLE object. CDO applications can access an attachment using the Attachment object.

authenticated user

A messaging user that has a valid account on a Microsoft Exchange server and can therefore access a mailbox. Also referred to as a validated user.

Automation

A Microsoft technology that allows objects to expose their internal services to each other as well as to human users, thereby permitting tasks to be performed automatically instead of by hand. Automation follows the Component Object Model (COM), and most Automation applications derive their objects from the **IDispatch** interface. Objects exposed through Automation include ActiveX objects, and applications that access them include ActiveX clients. Formerly referred to as OLE Automation.

Automation controller

A programming tool, such as Microsoft Visual Basic, that supports Microsoft Automation. An application written in or using an Automation controller can reference an arbitrary number of object libraries and access their objects from a single program.

B

browser

A client application connected to the World Wide Web that requests resources from a Web server, usually for the purpose of displaying them. An example of a browser is the Microsoft Internet Explorer. Also referred to as a Web browser.

C

calendar view

A view specifying a calendar rendering of a container object containing a collection of appointments. CDO applications can access a calendar view using the CalendarView object.

categorized view

See grouped view.

CDO

(Collaboration Data Objects) A technology for building messaging or collaboration applications. In versions previous to 1.1, CDO was called OLE Messaging; in version 1.1 it was called Active Messaging. It is designed to simplify the creation of applications with messaging functionality, or to add messaging functionality to existing applications.

CDO libraries

The set of programmable object libraries that expose messaging-related objects for use by an Automation controller. An application written in a tool supporting Automation can reference several object libraries and access all their objects in a single program. The CDO libraries are the CDO Library and the CDO Rendering Library.

CDO Library

An Automation programming interface that exposes programmable MAPI objects to a messaging user application. For more information, see the CDO Library Introduction and Overview.

CDO Rendering Library

An Automation programming interface that exposes programmable HTML rendering objects to a server-side script running on a Web server. For more information, see Overview of CDO Rendering.

child folder

A folder that is a child object of another folder, which is the child folder's parent folder. Also referred to as a subfolder.

child object

An object derived from another object, which is referred to as the parent object.

client-side script

Script that is decoded and run at a browser.

Collaboration Data Objects

See CDO.

collection

An object that contains zero or more objects of the same class. The CDO Library supports large collections and small collections. The CDO Rendering Library uses only small collections. Also referred to as a collection object or an object collection. See *also* container object.

collection object

See collection.

column

A vertical section of a table view that specifies rendering for one property on the contents of the container object being rendered. CDO applications can access a column using the Column object.

COM

(Component Object Model) An architecture for defining interfaces and interaction among objects implemented by widely varying software applications. A COM object instantiates one or more interfaces, each of which exposes zero or more properties and zero or more methods. All COM interfaces are derived from the base class **IUnknown**. Technologies built on the COM foundation include ActiveX, MAPI, and OLE.

common view

A predefined, persistent table view defined globally for all messaging users and all folders.

container object

An object that contains a collection of objects of one or more related classes. For example, a folder can contain both messages and child folders. See *also* object renderer.

container renderer

A rendering object used to render the contents of a CDO container object in tabular format. CDO applications can access a container renderer using the ContainerRenderer object.

conversation

A series of messages that pertain to the same topic. All the messages in a conversation typically have the same subject. Also referred to as a thread.

custom view

A nonpersistent table view created individually by a particular application, which applies only to one folder and one messaging user.

D

display name

A name associated with an object and used as a display token for that object. Many CDO objects have display names, exposed in each object's **Name** property. The display name of an AddressEntry object commonly contains a human user's friendly name or e-mail alias.

distribution list

(DL) An address entry representing a group of one or more address entries. A distribution list can contain individual messaging users and other distribution lists.

DL

See distribution list.

F

field

An object providing access to a MAPI property on a CDO Library object. CDO applications can access a field using the Field object.

File Transfer Protocol

See FTP.

folder

A container object that holds messages and other folders. CDO applications can access a folder using the Folder object. See *also* child folder, parent folder, personal folder, public folder.

folder view

A predefined, persistent table view defined individually for a particular folder.

format

An object specifying rendering information for exactly one property on an object being rendered. CDO applications can access a format using the Format object.

frame

A partition of a computer screen used by a browser to display images from HTML. Unless the HTML sent to the browser includes <FRAMESET> and <FRAME> tags, the display uses a single frame occupying the entire screen.

FTP

(File Transfer Protocol) A client/server protocol used on the World Wide Web to transfer a file from a server to a client. FTP is based on the TCP/IP protocol.

G

GAL

See global address list.

global address list

(GAL) An address book container that holds recipient entries for an entire organization and is available to all messaging users in that organization. A global address list is typically not modifiable by the users. See *also* personal address book.

group header

A nonpersistent object representing the display header for a grouping of messages in a grouped view on a folder. CDO applications can access a group header using the GroupHeader object.

grouped view

A table view specifying that the contents of the container object are to be sorted into groups for rendering. The sort is based on a specified property on the container object's contents. Each group is rendered along with its group header. Also referred to as a categorized view.

H

heading row

A row of an [HTML rendering](#) of a [table view](#) that contains the name of each [column](#) in the view. It can optionally be included in any desired [frame](#) of the rendering.

HTML

(Hypertext Markup Language) A tag language for representing documents with [hypertext](#) links. An HTML tag consists of a directive, possibly extended with one or more attributes, within angle brackets, for example . HTML is based on Standard Generalized Markup Language (SGML).

HTML rendering

The process or result of generating displayable [HTML](#) output from [objects](#) and [properties](#) of the [CDO Library](#). HTML rendering is accomplished using the [CDO Rendering Library](#). See *also* [rendering](#).

HTTP

(Hypertext Transfer Protocol) A client/server protocol used on the [World Wide Web](#) for sending and receiving [HTML](#) documents. HTTP is based on the [TCP/IP](#) protocol.

HTTP output

A stream containing text and [HTML](#) tags for the current displayable page, to be sent by [HTTP](#) to a [browser](#) either when the **End** method on the [response object](#) is called or when the current [script](#) finishes executing.

hypertext

A collection of documents containing cross-reference links that can be used interactively by a user to move immediately from one topic to another.

Hypertext Markup Language

See [HTML](#).

Hypertext Transfer Protocol

See [HTTP](#).

I

IIS

(Internet Information Server) A Web server integrated into Windows NT server. Microsoft IIS is required to access applications based on Microsoft ASP.

impersonation

Associating a set of Windows NT security credentials with an execution thread. This enables the thread to log on to a session.

interface

The definition of a class of objects of similar behavior. An object is an instance of one or more interfaces. The COM architecture is the foundation for Microsoft interfaces.

Internet

A worldwide hierarchy of computer networks using a variety of protocols. At its highest level the Internet is connected by backbone networks such as ARPANet, NSFNet, and MILNET. The backbones connect transit networks, which in turn connect stub networks. Logically, Internet participants are represented by a domain such as .com, .org, and .edu, by a logical network within the domain, and by a server within the logical network. An example of a logical address is "www.microsoft.com".

Internet Information Server

See IIS.

Internet Server Application Programming Interface

See ISAPI.

IPM subtree

The hierarchy of folders for all interpersonal messages. An interpersonal message is sent or received by human users rather than applications or processes. It has a message class that starts with IPM, such as IPM.Note.

ISAPI

(Internet Server Application Programming Interface) A Microsoft interface for writing in-process extensions to IIS. ASP is an ISAPI application.

L

large collection

A collection for which the service provider cannot always maintain an accurate count of member objects. Large collections support **Get** methods that enable you to access individual members of the collection. Currently, the large collections are the AddressEntries, Folders, and Messages collections.

M

mailbox

The set of folders held in the IPM subtree of a messaging user. The mailbox is intended for interpersonal messages. Its folders typically include the Inbox, Outbox, Sent Items, and Deleted Items.

MAPI

(Messaging Application Programming Interface) A messaging architecture enabling multiple applications to interact with multiple messaging systems across a variety of hardware platforms. MAPI is built on the COM foundation.

message

An object containing information that is sent from a sender to one or more recipients or that is posted in a public folder. CDO applications can access a message using the Message object.

message store

A container object that holds folders organized hierarchically. CDO applications can access a message store using the InfoStore object.

messaging user

An object that is capable of sending or receiving messages. A messaging user can be a human user or an application or process. See *also* authenticated user, unauthenticated user.

method

A procedure that is exposed by an object and performs a specific action.

O

object

A programmable software component representing an instance of one or more defined interfaces. The objects exposed by the CDO libraries conform to the COM architecture and expose zero or more properties and zero or more methods.

object collection

See collection.

object renderer

A rendering object used to render one or more selected properties on a CDO object, rather than the entire object. CDO applications can access an object renderer using the ObjectRenderer object. See *also* container renderer.

OCX

(OLE Custom Controls) See ActiveX control.

OLE

A software technology built on the COM foundation and used for creating and working with compound documents. OLE objects commonly use the **IStream** and **IStorage** interfaces. See *also* ActiveX.

OLE Automation

See Automation.

OLE Messaging

See CDO.

P

PAB

See [personal address book](#).

parent folder

A [folder](#) that is the [parent object](#) of another folder, which is the parent folder's [child folder](#).

parent object

An [object](#) from which another object is derived. The derived object is a [child object](#).

pattern

An [object](#) specifying [rendering](#) information for a particular set of values of a [property](#) on an object being rendered. CDO applications can access a pattern using the [Pattern](#) object.

PDL

See [private distribution list](#).

personal address book

(PAB) A modifiable [address book container](#) that holds [recipient](#) entries either created by the [messaging user](#) or copied from other address book containers such as a [global address list](#). See also [global address list](#).

personal folder

A [folder](#) held outside of the [mailbox](#) of a [messaging user](#), in which the user can store selected [messages](#). A messaging user's personal folders are normally held in the user's personal [message store](#).

personal view

A predefined [table view](#) defined individually for a particular [messaging user](#).

personal Web server

A [Web server](#) that does not use [IIS](#). Because it does not require a Windows server, a personal Web server can run on a Windows workstation.

private distribution list

(PDL) A [distribution list](#) that exists only in a [personal address book](#) belonging to one [messaging user](#). A private distribution list can contain individual messaging users, distribution lists, and other private distribution lists. However, a PDL does not have an e-mail address. For more information, see the [DisplayType](#) property of the [AddressEntry](#) object.

profile

Configuration information about the set of message services for a [session](#). Profiles are created from information stored in the [MAPI](#) configuration file, MAPISVC.INF. A profile indicates which [address books](#) and [message stores](#) are accessible to the [messaging user](#) that is logging on, as well as the messaging user's own [display name](#) and addressing information.

property

A data attribute exposed by an [object](#). A property represents data that is held inside the object and is inaccessible from outside except through the object's defined [interface](#). An object may permit read/write access to some of its properties and read-only access to others.

public folder

A folder held outside of the mailboxes of all the messaging users on a message store, in which any connected user can post selected messages. A public folder can be used as a bulletin board or online forum.

R

recipient

A messaging user or distribution list designated to receive a particular message. Recipients are usually held in address book containers. CDO applications can access a recipient using the Recipient object.

renderable object

A CDO object being used as a data source for a rendering object.

renderable property

A property on a CDO object being used as a data source for a rendering object.

rendering

The process or result of preparing an image for display. The image can include text, graphics, and form elements such as frames and borders. Rendering converts coded representations of the components of the image into a single output suitable for displaying. See *a/so* HTML rendering.

rendering object

A CDO Rendering object used for rendering a CDO object, or one or more of its properties, into HTML output. Currently, the rendering objects are the container renderer and object renderer objects.

rendering source

A string providing information that a rendering object uses to render a particular property or value into hypertext. A rendering source contains HTML tags and substitution tokens.

resolution

The process of associating a valid address with a display name or addressing information. The names of all recipients for a message must be resolved before the message can be sent.

response object

An Active Server™ object used to send HTML output to a browser. A response object implements the **IResponse** interface, which controls page buffering, cookie values, Web server logging, and browser page caching.

S

script

Code that is run in an application for a special purpose. Script can be in any scripting language.

scripting language

Any language, such as Visual Basic Scripting Edition (VBScript) or JavaScript (JScript) that can compile or interpret special script.

server-side script

Script that is decoded and run at a Web server.

session

An active connection between a client application and the MAPI subsystem. A session is begun when a messaging user logs on to the system and references a profile. The profile determines the available messaging operations and the service providers available to handle the operations. CDO applications can access a session using the Session object.

small collection

A collection for which the service provider maintains an accurate count of member objects. You can directly access individual members of the collection using an index. Currently, the large collections are the AddressLists, Attachments, Columns, Fields, Formats, InfoStores, Patterns, Recipients, and Views collections.

subfolder

A child folder contained in a parent folder.

T

table view

A view specifying a tabular rendering of a container object. CDO applications can access a table view using the TableView object. See also common view, custom view, folder view, and personal view.

TCP/IP

(Transmission Control Protocol over Internet Protocol) One of the most commonly used protocol suites on the Internet. IP is a network layer protocol that handles packet switching, fragmentation, and routing. TCP is a transport layer protocol built on top of IP and handles flow control, multiplexing, and error control.

top-level object

A CDO libraries object that can be defined directly from a program and does not have to be derived from another already-defined object. Currently, the top-level objects are the CDO Session object and the CDO Rendering ContainerRenderer, ObjectRenderer, and RenderingApplication objects. For more information, see Top-Level Objects and CDO Rendering Objects.

transport

A service provider responsible for transferring messages between a message store and an underlying messaging system that delivers the messages.

U

unauthenticated user

A messaging user that does not have a recognized account on any Microsoft Exchange server. An unauthenticated user can log on to a server anonymously, but is restricted to accessing only the published public folders and address books. Also referred to as an anonymous user.

uniform resource locator

See URL.

URL

(Uniform Resource Locator) A standardized string used to specify a resource on the Internet, such as an HTML document. The format of a URL is "protocol://server.network.domain/path/resource".

V

validated user

See [authenticated user](#).

VBX

(Visual Basic Extensions) See [ActiveX control](#).

view

An [object](#) specifying a particular type of [rendering](#) of a [container object](#). Currently, the [CDO Rendering Library](#) supports calendar views and [table views](#).

virtual directory

See [virtual root](#).

virtual root

A disk directory on [IIS](#) designated as a share point, which can be used as a root directory for paths to its subdirectories. Also referred to as a virtual directory.

Visual InterDev

A component of Microsoft® Visual Studio™ that serves as the development platform for applications dealing with the [World Wide Web](#). Microsoft® Visual InterDev™ supports creation and editing of .HTM and .ASP files, and development of [scripts](#) in scripting languages such as VBScript and JScript.

W

W3C

(World Wide Web Consortium) The international standards group for the World Wide Web (WWW or W3), funded by its industrial members but operated principally by the Massachusetts Institute of Technology (MIT), the Institut National de Recherche en Informatique et Automatique (INRIA), and the Center for European Particle Research (CERN).

Web

See World Wide Web.

Web browser

See browser.

Web server

A program connected to the World Wide Web that furnishes resources upon request from a browser. The requested resource is usually identified by a URL.

World Wide Web

(WWW, W3) A distributed client/server information retrieval system using multiple protocols on the Internet. The client is a browser and the server is a Web server. Typical protocols include HTTP, FTP, and Gopher. Also referred to as the Web or the WWW.

WWW

See World Wide Web.

X

X.400

An international message-handling standard for connecting e-mail networks to each other and to messaging users, published by the International Telecommunications Union (ITU).

X.500

An international message-handling standard for directory services (DS), published by the International Telecommunications Union (ITU).

