[MS-OXCFOLD]:
Folder Object Protocol Specification

Revision Summary

Date	Revision History	Revision Class	Comments
04/04/2008	0.1		Initial Availability.
04/25/2008	0.2		Revised and updated property names and other technical content.
06/27/2008	1.0		Initial Release.
08/06/2008	1.01		Revised and edited technical content.
09/03/2008	1.02		Revised and edited technical content.
12/03/2008	1.03		Revised and edited technical content.
03/04/2009	1.04		Revised and edited technical content.
04/10/2009	2.0		Updated technical content and applicable product releases.
07/15/2009	3.0	Major	Revised and edited for technical content.
11/04/2009	4.0.0	Major	Updated and revised the technical content.
02/10/2010	5.0.0	Major	Updated and revised the technical content.
05/05/2010	5.0.1	Editorial	Revised and edited the technical content.
08/04/2010	6.0	Major	Significantly changed the technical content.

Contents

1 Introduction 9
1.1 Glossary 9
1.2 References 10
1.2.1 Normative References 10
1.2.2 Informative References 11
1.3 Overview 11
1.3.1 Manipulation of Folder objects 11
1.4 Relationship to Other Protocols 11
1.5 Prerequisites/Preconditions 12
1.6 Applicability Statement 12
1.7 Versioning and Capability Negotiation 12
1.8 Vendor-Extensible Fields 12
1.9 Standards Assignments 12

2 Messages 13
2.1 Transport 13
2.2 Message Syntax 13
2.2.1 RopOpenFolder 13
2.2.1.1 Request Parameter Overview 13
2.2.1.1.1 InputHandleIndex 13
2.2.1.1.2 OutputHandleIndex 13
2.2.1.1.3 FolderId 14
2.2.1.1.4 OpenModeFlags 14
2.2.1.2 Response Parameter Overview 14
2.2.1.2.1 ReturnValue 14
2.2.1.2.2 HasRules 14
2.2.1.2.3 IsGhosted 14
2.2.1.2.4 ServerCount 14
2.2.1.2.5 CheapServerCount 14
2.2.1.2.6 Servers 15
2.2.2 RopCreateFolder 15
2.2.2.1 Request Parameter Overview 15
2.2.2.1.1 InputHandleIndex 15
2.2.2.1.2 OutputHandleIndex 15
2.2.2.1.3 FolderType 15
2.2.2.1.4 UseUnicodeStrings 15
2.2.2.1.5 OpenExisting 16
2.2.2.1.6 Reserved 16
2.2.2.1.7 DisplayName 16
2.2.2.1.8 Comment 16
2.2.2.2 Response Parameter Overview 16
2.2.2.2.1 ReturnValue 16
2.2.2.2.2 FolderId 16
2.2.2.2.3 IsExistingFolder 16
2.2.2.2.4 HasRules 16
2.2.2.2.5 IsGhosted 16
2.2.2.2.6 ServerCount 17
2.2.2.2.7 CheapServerCount 17
2.2.2.2.8 Servers 17
2.2.3 RopDeleteFolder 17

2.2.3.1 Request Parameter Overview 17
2.2.3.1.1 InputHandleIndex 17
2.2.3.1.2 DeleteFolderFlags 17
2.2.3.1.3 FolderId 18
2.2.3.2 Response Parameter Overview 18
2.2.3.2.1 ReturnValue 18
2.2.3.2.2 PartialCompletion 18
2.2.4 RopSetSearchCriteria 18
2.2.4.1 Request Parameter Overview 18
2.2.4.1.1 InputHandleIndex 18
2.2.4.1.2 RestrictionDataSize 18
2.2.4.1.3 RestrictionData 19
2.2.4.1.4 FolderIdCount 19
2.2.4.1.5 FolderIds 19
2.2.4.1.6 SearchFlags 19
2.2.4.2 Response Parameter Overview 20
2.2.4.2.1 ReturnValue 20
2.2.5 RopGetSearchCriteria 20
2.2.5.1 Request Parameter Overview 20
2.2.5.1.1 InputHandleIndex 21
2.2.5.1.2 UseUnicode 21
2.2.5.1.3 IncludeRestriction 21
2.2.5.1.4 IncludeFolders 21
2.2.5.2 Response Parameter Overview 21
2.2.5.2.1 ReturnValue 21
2.2.5.2.2 RestrictionDataSize 21
2.2.5.2.3 RestrictionData 21
2.2.5.2.4 FolderIdCount 21
2.2.5.2.5 FolderIds 21
2.2.5.2.6 SearchFlags 21
2.2.6 RopMoveCopyMessages 22
2.2.6.1 Request Parameter Overview 23
2.2.6.1.1 SourceHandleIndex 23
2.2.6.1.2 DestHandleIndex 23
2.2.6.1.3 MessageIdCount 23
2.2.6.1.4 MessageIds 23
2.2.6.1.5 WantAsynchronous 23
2.2.6.1.6 WantCopy 23
2.2.6.2 Response Parameter Overview 23
2.2.6.2.1 ReturnValue 23
2.2.6.2.2 PartialCompletion 23
2.2.6.3 Null Destination Failure Response Parameter Overview 24
2.2.6.3.1 ReturnValue 24
2.2.6.3.2 PartialCompletion 24
2.2.7 RopMoveFolder 24
2.2.7.1 Request Parameter Overview 24
2.2.7.1.1 SourceHandleIndex 24
2.2.7.1.2 DestHandleIndex 24
2.2.7.1.3 WantAsynchronous 24
2.2.7.1.4 UseUnicode 25
2.2.7.1.5 FolderId 25
2.2.7.1.6 NewFolderName 25
2.2.7.2 Response Parameter Overview 25

2.2.7.2.1 ReturnValue 25
2.2.7.2.2 PartialCompletion 25
2.2.7.3 Null Destination Failure Response Parameter Overview 25
2.2.7.3.1 ReturnValue 25
2.2.7.3.2 PartialCompletion 25
2.2.8 RopCopyFolder 25
2.2.8.1 Request Parameter Overview 26
2.2.8.1.1 SourceHandleIndex 26
2.2.8.1.2 DestHandleIndex 26
2.2.8.1.3 WantAsynchronous 26
2.2.8.1.4 WantRecursive 26
2.2.8.1.5 UseUnicode 26
2.2.8.1.6 FolderId 26
2.2.8.1.7 NewFolderName 26
2.2.8.2 Response Parameter Overview 26
2.2.8.2.1 ReturnValue 26
2.2.8.2.2 PartialCompletion 27
2.2.8.3 Null Destination Failure Response Parameter Overview 27
2.2.8.3.1 ReturnValue 27
2.2.8.3.2 PartialCompletion 27
2.2.9 RopEmptyFolder 27
2.2.9.1 Request Parameter Overview 27
2.2.9.1.1 InputHandleIndex 27
2.2.9.1.2 WantAsynchronous 27
2.2.9.1.3 WantDeleteAssociated 28
2.2.9.2 Response Parameter Overview 28
2.2.9.2.1 ReturnValue 28
2.2.9.2.2 PartialCompletion 28
2.2.10 RopHardDeleteMessagesAndSubfolders 28
2.2.10.1 Request Parameter Overview 28
2.2.10.1.1 InputHandleIndex 28
2.2.10.1.2 WantAsynchronous 28
2.2.10.1.3 WantDeleteAssociated 29
2.2.10.2 Response Parameter Overview 29
2.2.10.2.1 ReturnValue 29
2.2.10.2.2 PartialCompletion 29
2.2.11 RopDeleteMessages 29
2.2.11.1 Request Parameter Overview 29
2.2.11.1.1 InputHandleIndex 29
2.2.11.1.2 WantAsynchronous 29
2.2.11.1.3 NotifyNonRead 29
2.2.11.1.4 MessageIdCount 30
2.2.11.1.5 MessageIds 30
2.2.11.2 Response Parameter Overview 30
2.2.11.2.1 ReturnValue 30
2.2.11.2.2 PartialCompletion 30
2.2.12 RopHardDeleteMessages 30
2.2.12.1 Request Parameter Overview 30
2.2.12.1.1 InputHandleIndex 30
2.2.12.1.2 WantAsynchronous 30
2.2.12.1.3 NotifyNonRead 31
2.2.12.1.4 MessageIdCount 31
2.2.12.1.5 MessageIds 31

2.2.12.2 Response Parameter Overview 31
2.2.12.2.1 ReturnValue 31
2.2.12.2.2 PartialCompletion 31
2.2.13 RopGetHierarchyTable 31
2.2.13.1 Request Parameter Overview 31
2.2.13.1.1 InputHandleIndex 31
2.2.13.1.2 OutputHandleIndex 32
2.2.13.1.3 TableFlags 32
2.2.13.2 Response Parameter Overview 32
2.2.13.2.1 ReturnValue 32
2.2.13.2.2 RowCount 32
2.2.14 RopGetContentsTable 32
2.2.14.1 Request Parameter Overview 33
2.2.14.1.1 InputHandleIndex 33
2.2.14.1.2 OutputHandleIndex 33
2.2.14.1.3 TableFlags 33
2.2.14.2 Response Parameter Overview 34
2.2.14.2.1 ReturnValue 34
2.2.14.2.2 RowCount 34
2.3 Folder Object Properties 34
2.3.1 General Properties 34
2.3.2 Folder Object Specific Properties 34
2.3.2.1 Read-Only Properties 35
2.3.2.1.1 PidTagContentCount 35
2.3.2.1.2 PidTagContentUnreadCount 35
2.3.2.1.3 PidTagDeletedOn 35
2.3.2.1.4 PidTagAddressBookEntryId 35
2.3.2.1.5 PidTagFolderId 35
2.3.2.1.6 PidTagHierarchyChangeNumber 35
2.3.2.1.7 PidTagMessageSize 35
2.3.2.1.8 PidTagMessageSizeExtended 35
2.3.2.1.9 PidTagSubfolders 35
2.3.2.2 Read/Write Properties 35
2.3.2.2.1 PidTagAttributeHidden 35
2.3.2.2.2 PidTagComment 36
2.3.2.2.3 PidTagDisplayName 36
2.3.2.2.4 PidTagFolderType 36
2.3.2.2.5 PidTagRights 36

3 Protocol Details 37
3.1 Client Details 37
3.1.1 Abstract Data Model 37
3.1.1.1 Hierarchy Table 37
3.1.1.2 Contents Table 37
3.1.2 Timers 37
3.1.3 Initialization 38
3.1.4 Higher-Layer Triggered Events 38
3.1.4.1 Open a Folder 38
3.1.4.2 Create a Folder 38
3.1.4.3 Delete a Folder 38
3.1.4.4 Set Search Criteria 38
3.1.4.5 Get Search Criteria 39
3.1.4.6 Move or Copy Messages 39

3.1.4.7 Move Folder 39
3.1.4.8 Copy Folder 39
3.1.4.9 Empty a Folder 39
3.1.4.10 Delete Messages 39
3.1.4.11 Hierarchy Table 40
3.1.4.12 Contents Table 40
3.1.5 Message Processing Events and Sequencing Rules 40
3.1.6 Timer Events 40
3.1.7 Other Local Events 40
3.2 Server Details 41
3.2.1 Abstract Data Model 41
3.2.2 Timers 41
3.2.3 Initialization 41
3.2.4 Higher-Layer Triggered Events 41
3.2.5 Message Processing Events and Sequencing Rules 41
3.2.5.1 RopOpenFolder 41
3.2.5.2 RopCreateFolder 42
3.2.5.3 RopDeleteFolder 42
3.2.5.4 RopSetSearchCriteria 43
3.2.5.5 RopGetSearchCriteria 45
3.2.5.6 RopMoveCopyMessages 45
3.2.5.7 RopMoveFolder 46
3.2.5.8 RopCopyFolder 46
3.2.5.9 RopEmptyFolder 46
3.2.5.10 RopHardDeleteMessagesAndSubfolders 47
3.2.5.11 RopDeleteMessages 47
3.2.5.12 RopHardDeleteMessages 47
3.2.5.13 RopGetHierarchyTable 48
3.2.5.14 RopGetContentsTable 48
3.2.6 Timer Events 49
3.2.7 Other Local Events 49

4 Protocol Examples 50
4.1 Creating a New Folder by Using RopCreateFolder 50
4.1.1 Client Request Buffer 50
4.1.2 Server Responds to Client Request 51
4.2 Deleting an Existing Folder by Using RopDeleteFolder 52
4.2.1 Client Request Buffer 52
4.2.2 Server Responds to Client Request 52
4.3 Deleting Messages Within a Folder 53
4.3.1 Client Request Buffer 53
4.3.2 Server Responds to Client Request 54
4.4 Moving Messages From One Folder to Another 54
4.4.1 Client Request Buffer 54
4.4.2 Server Responds to Client Request 55
4.5 Moving a Folder 55
4.5.1 Client Request Buffer 55
4.5.2 Server Responds to Client Request 56
4.6 Copying a Folder 56
4.6.1 Client Request Buffer 57
4.6.2 Server Responds to Client Request 57
4.7 Getting the List of Subfolders Within a Message Folder 58
4.7.1 Client Request Buffer 58

4.7.2 Server Responds to Client Request 58
4.8 Setting the Search Criteria for a Search Folder 59
4.8.1 Client Request Buffer 59
4.8.2 Server Responds to Client Request 62
4.9 Getting the Search Criteria for a Search Folder 63
4.9.1 Client Request Buffer 63
4.9.2 Server Responds to Client Request 63

5 Security 68
5.1 Security Considerations for Implementers 68
5.2 Index of Security Parameters 68

6 Appendix A: Product Behavior 69

7 Change Tracking 70

8 Index 73

1 Introduction

A **folder** is a **messaging object** that serves as the basic unit of organization for **messages**. Folder operations provide a way to manipulate folder **properties** and messages inside the folder.

This document specifies the following:

- **Folder objects**
- The **remote operations (ROPs)** that are available to manipulate Folder objects
- The behavior of ROPs and their parameter descriptions
- General folder properties

The Folder Object protocol uses ROPs as a transport protocol between the client and the server. This specification assumes that the reader is familiar with the ROP concepts and requirements that are specified in [MS-OXCROPS]. Those concepts and requirements are not repeated in this specification.

1.1 Glossary

The following terms are defined in [MS-OXGLOS]:

active replica
contents table
EntryID
folder
folder associated information (FAI)
folder ID (FID)
Folder object
ghosted folder
handle
hard delete
hierarchy table
little-endian
logon object
mailbox
message
message ID (MID)
Message object
messaging object
normal message
permissions
property (1)
property type
public folder
read receipt
remote operation (ROP)
remote procedure call (RPC)
replica (1)
restriction
Root folder
ROP request buffer
ROP response buffer
rule

search criteria
search folder
search folder container
Server object
Server object handle
Server object handle table
soft delete
special folder
store
stream (1)
table
Table object
Unicode

The following terms are specific to this document:

full-text index: A digitally stored list of search terms culled from examining the entire contents of a body of documents, **messages**, or other text objects, in order to increase the speed of search results.

full-text search: In text retrieval, a technique for searching a computer-stored document or database by examining all the words in every stored document, in an attempt to match search words supplied by the client.

sibling folder: The name given to two or more generic **folders** that have the same parent **folder**.

MAY, SHOULD, MUST, SHOULD NOT, MUST NOT: These terms (in all caps) are used as described in [RFC2119]. All statements of optional behavior use either MAY, SHOULD, or SHOULD NOT.

1.2 References

1.2.1 Normative References

We conduct frequent surveys of the normative references to assure their continued availability. If you have any issue with finding a normative reference, please contact dochelp@microsoft.com. We will assist you in finding the relevant information. Please check the archive site, http://msdn2.microsoft.com/en-us/library/E4BD6494-06AD-4aed-9823-445E921C9624, as an additional source.

[MS-OXCDATA] Microsoft Corporation, "Data Structures", April 2008.

[MS-OXCMSG] Microsoft Corporation, "Message and Attachment Object Protocol Specification", April 2008.

[MS-OXCNOTIF] Microsoft Corporation, "Core Notifications Protocol Specification", April 2008.

[MS-OXCPERM] Microsoft Corporation, "Exchange Access and Operation Permissions Protocol Specification", April 2008.

[MS-OXCPRPT] Microsoft Corporation, "Property and Stream Object Protocol Specification", April 2008.

[MS-OXCROPS] Microsoft Corporation, "Remote Operations (ROP) List and Encoding Protocol Specification", April 2008.

[MS-OXCRPC] Microsoft Corporation, "Wire Format Protocol Specification", April 2008.

[MS-OXCSTOR] Microsoft Corporation, "Store Object Protocol Specification", April 2008.

[MS-OXCTABL] Microsoft Corporation, "Table Object Protocol Specification", April 2008.

[MS-OXOMSG] Microsoft Corporation, "E-Mail Object Protocol Specification", April 2008.

[MS-OXORULE] Microsoft Corporation, "E-Mail Rules Protocol Specification", April 2008.

[MS-OXOSFLD] Microsoft Corporation, "Special Folders Protocol Specification", April 2008.

[MS-OXOSRCH] Microsoft Corporation, "Search Folder List Configuration Protocol Specification", April 2008.

[MS-OXPROPS] Microsoft Corporation, "Exchange Server Protocols Master Property List", April 2008.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997, http://www.ietf.org/rfc/rfc2119.txt

1.2.2 Informative References

[MS-OXGLOS] Microsoft Corporation, "Exchange Server Protocols Master Glossary", April 2008.

1.3 Overview

A folder is an object in a message **store** that serves as the basic unit of organization for messages. Folders are arranged hierarchically, and contain properties, messages, **folder associated information (FAI)**, and other folders.

The following are the three types of folders:

- **Root folders**. Every messagestore has a Root folder. The Root folder appears at the top of the folder hierarchy, and can contain properties, messages, and other folders. Root folders cannot be moved, copied, renamed, or deleted. There is only one Root folder for each messagestore.
- Generic folders. Like Root folders, generic folders contain messages and other folders. Unlike Root folders, they can be moved, copied, renamed, and deleted. Generic folders can be created within the Root folder or other generic folders. The folder in which the new folder is created is referred to as the parent folder of the new folder. Generic folders that have the same parent are called **sibling folders**.
- **Search folders**. A search folder contains a list of references to messages that are compiled by the server according to a set of criteria given to the folder. Therefore, a search folder cannot contain any real objects. Any operation on a message that is referenced in a search folder is performed on the message in the folder that actually contains it. For more details about search folders, including usages, **restrictions**, and notes, see [MS-OXOSRCH].

1.3.1 Manipulation of Folder objects

Clients send remote operations (ROPs) to the server to create, copy, move, or delete folders, to copy, delete, or move messages, and to modify folder **permissions**. For more details about folder permissions, see [MS-OXCPERM].

1.4 Relationship to Other Protocols

The Folder Object protocol depends on the following:

- Messages, **tables**, and properties, as specified in [MS-OXCMSG], [MS-OXOMSG], [MS-OXOSFLD], [MS-OXCTABL], [MS-OXCPRPT], and [MS-OXPROPS].
- The underlying remote operation (ROP) transport, as specified in [MS-OXCROPS].
- The message store, as specified in [MS-OXCSTOR].
- The ability to manipulate tables in the message store, as specified in [MS-OXCTABL] and [MS-OXCNOTIF].
- The ability to set permissions on folders, as specified in [MS-OXCPERM].

The following protocols extend the Folder Object protocol:

- Search Folder List Configuration protocol, as specified in [MS-OXOSRCH].
- Special Folders protocol, as specified in [MS-OXOSFLD].

1.5 Prerequisites/Preconditions

This specification assumes that the messaging client has previously logged on to the server and has acquired a **handle** to the object on which it is going to operate. Methods to open the object and acquire a handle are dependent on the object type and are specified in [MS-OXCSTOR] for stores, and [MS-OXCMSG] for messages.

1.6 Applicability Statement

This protocol provides a hierarchical organization model for messages in a store.

1.7 Versioning and Capability Negotiation

None.

1.8 Vendor-Extensible Fields

None.

1.9 Standards Assignments

None.

2 Messages

2.1 Transport

The **ROP request buffers** and **ROP response buffers** that are specified by this protocol are sent to, and received from, the server respectively by using the underlying Wire Format protocol, as specified in [MS-OXCRPC].

2.2 Message Syntax

Folder objects can be created and modified by clients and servers. Except where noted, this section defines constraints under which both clients and servers operate when creating and modifying Folder objects.

The following sections specify the format of ROP request buffers and response buffers that are specific to folder operations.<1> Before sending these requests to the server, the client needs to be logged on to the server, and needs to open or acquire handles to the messaging objects that are used in the ROP requests. For more details about logging on to the server, including usages, restrictions, and notes, see [MS-OXCSTOR]. Also, ROPs that require a **folder ID (FID)** or **message ID (MID)** need to acquire those IDs for the objects to be used in the ROP requests. For more details about acquiring MIDs, including usages, restrictions, and notes, see [MS-OXCMSG].

The request buffers and response buffers that are specified in this section do not include the *RopId* and *LogonID* parameters that are included as the first two bytes of every ROP request buffer. For details about the *RopId* and *LogonID* parameters, see [MS-OXCROPS] section 2.2.4.

2.2.1 RopOpenFolder

The RopOpenFolder operation opens an existing folder.

The client application MUST send a RopRelease request after executing all subsequent operations on the opened folder as specified in [MS-OXCROPS] section 2.2.15.3.1.

The complete syntax of the RopOpenFolder request and response buffers is specified in [MS-OXCROPS] section 2.2.4.1. This section specifies the syntax and semantics of various fields that are not fully specified in [MS-OXCROPS].

2.2.1.1 Request Parameter Overview

2.2.1.1.1 InputHandleIndex

The *InputHandleIndex* parameter specifies the location in the **Server object handle table** where the handle for the input handle is stored. The input handle for this operation is a **logon object** handle or a Folder object handle. For more information about logon objects, see [MS-OXCSTOR] section 1.6.

2.2.1.1.2 OutputHandleIndex

The *OutputHandleIndex* parameter specifies the location in the Server object handle table where the handle for the output handle is stored. The output handle for this operation is a Folder object handle.

2.2.1.1.3 FolderId

The *FolderId* parameter contains the folder ID (FID) of the folder to be opened. The FID structure is defined in [MS-OXCDATA] section 2.2.1.1.

2.2.1.1.4 OpenModeFlags

The *OpenModeFlags* parameter contains a bitmask of flags that indicate the open folder mode.

The following table specifies the valid values for this flag. All other bits MUST NOT be set by the client, and MUST be ignored by the server.

Name	Value	Description
OpenSoftDeleted	0x04	If this bit is not set, then it indicates the opening of an existing folder. If this bit is set, then it indicates the opening of either an existing or **soft deleted** folder.

2.2.1.2 Response Parameter Overview

2.2.1.2.1 ReturnValue

The *ReturnValue* parameter indicates the result of the operation. The server returns "0x00000000" to indicate success. For more details about common error codes, see [MS-OXCDATA] section 2.4.

2.2.1.2.2 HasRules

If there are rules on the server associated with this folder, the server sets the *HasRules* parameter value to nonzero (TRUE). If no rules are associated with this folder, the *HasRules* value is set to 0x00 (FALSE).<2>

For more details about rules, see [MS-OXORULE].

2.2.1.2.3 IsGhosted

If the server does not host an **active replica** of the folder, the server sets the *IsGhosted* parameter value to nonzero, and the response buffer MUST contain the *ServerCount*, *CheapServerCount*, and *Servers* parameters. Otherwise, the *IsGhosted* parameter value is set to 0 (zero). The *IsGhosted* parameter is only present for folders in public stores.

For more details about **ghosted folders**, see RopPublicFolderIsGhosted in [MS-OXCSTOR] section 2.2.1.7.

2.2.1.2.4 ServerCount

The *ServerCount* parameter contains the number of servers that have a **replica** of the folder.

This field is only present if the *IsGhosted* field is nonzero (TRUE).

2.2.1.2.5 CheapServerCount

The *CheapServerCount* parameter contains the number of the cheapest, same-cost servers at the front of the server list.

This field is only present if the *IsGhosted* parameter is nonzero (TRUE).

For more details about the *CheapServerCount* parameter, see RopPublicFolderIsGhosted in [MS-OXCSTOR] section 2.2.1.7.2.3.

2.2.1.2.6 Servers

The *Servers* parameter contains a list of null-terminated strings that specify which servers have replicas of this folder.

This parameter is only present if the *IsGhosted* parameter is nonzero (TRUE).

For more details about the *Servers* parameter, see the RopPublicFolderIsGhosted in [MS-OXCSTOR] section 2.2.1.7.2.4.

2.2.2 RopCreateFolder

RopCreateFolder creates a new subfolder. This ROP creates either **public folders** or private **mailbox** folders.

The client application MUST send a RopRelease request after executing all subsequent operations on the created folder.

The complete syntax of the RopCreateFolder request and response buffers is specified in [MS-OXCROPS] section 2.2.4.2. This section specifies the syntax and semantics of various fields that are not fully specified in [MS-OXCROPS].

2.2.2.1 Request Parameter Overview

2.2.2.1.1 InputHandleIndex

The *InputHandleIndex* parameter specifies the location in the Server object handle table where the handle for the input handle is stored. The input handle for this operation is a Folder object handle that represents the parent folder of the folder to be created.

2.2.2.1.2 OutputHandleIndex

The *OutputHandleIndex* parameter specifies the location in the Server object handle table where the handle for the output handle is stored. The output handle for this operation is a Folder object handle.

2.2.2.1.3 FolderType

The *FolderType* parameter contains the type of folder to be created. One of the values specified in the following table MUST be used.

Value	Folder type
0x01	Generic folder.
0x02	Search folder

2.2.2.1.4 UseUnicodeStrings

The *UseUnicodeStrings* parameter value is nonzero (TRUE) if **DisplayName** and **Comment** are formatted in **Unicode**. Otherwise, the *UseUnicodeStrings* parameter value is set to 0x00 (FALSE).

2.2.2.1.5 OpenExisting

If the *OpenExisting* parameter value is set to nonzero (TRUE), a pre-existing folder whose name is identical to the name specified in the *DisplayName* parameter is opened. Otherwise, the request fails if a folder with an identical name already exists.

2.2.2.1.6 Reserved

Client applications MUST set this parameter to 0x00 (FALSE).

2.2.2.1.7 DisplayName

The *DisplayName* parameter contains a null-terminated folder display name string. This name becomes the value of the new folder's PidTagDisplayName property.

2.2.2.1.8 Comment

The *Comment* parameter contains a null-terminated folder comment string that is associated with the new folder. This string becomes the value of the new folder's PidTagComment property.

2.2.2.2 Response Parameter Overview

2.2.2.2.1 ReturnValue

The *ReturnValue* parameter indicates the result of the operation. The server returns "0x00000000" to indicate success. For more details about common error codes, see [MS-OXCDATA] section 2.4.

2.2.2.2.2 FolderId

The *FolderId* parameter contains the FID of the folder that was created or opened.

2.2.2.2.3 IsExistingFolder

If the name given by the *DisplayName* parameter (section 2.2.2.1.7) in the request buffer already exists, the server sets the value of the *IsExistingFolder* parameter to nonzero (TRUE). <3> If the folder does not exist, then the server sets the value of the *IsExistingFolder* parameter to 0x00 (FALSE).

2.2.2.2.4 HasRules

If there are rules on the server that are associated with this folder, the server sets the *HasRules* parameter value to nonzero (TRUE). If no rules are associated with this folder, the *HasRules* value is set to 0x00 (FALSE). The *HasRules* parameter is present only if *IsExistingFolder* parameter is nonzero (TRUE) and it is a public folder store.

For more details about rules, see [MS-OXORULE].

2.2.2.2.5 IsGhosted

If the server does not host an active replica of the folder, the server sets the *IsGhosted* parameter value to nonzero, and the response buffer MUST contain the *ServerCount*, *CheapServerCount*, and *Servers* parameters. Otherwise, the *IsGhosted* parameter value is set to 0 (zero).

For more details about ghosted folders, see RopPublicFolderIsGhosted in [MS-OXCSTOR] section 2.2.1.7.

2.2.2.2.6 ServerCount

The *ServerCount* parameter contains the number of servers that have a replica of the folder.

This field is only present if the *IsGhosted* parameter has a nonzero (TRUE) value.

2.2.2.2.7 CheapServerCount

The *CheapServerCount* parameter contains the number of the cheapest, same cost servers at the front of the server list. This value MUST be less than or equal to the *ServerCount* parameter (section 2.2.2.2.6), and MUST be greater than zero when *ServerCount* is greater than zero.

This field is only present if the *IsGhosted* parameter (section 2.2.2.2.5) has a nonzero (TRUE) value.

2.2.2.2.8 Servers

The *Servers* parameter contains a list of null-terminated strings that specify which servers have replicas of this folder.

This field is only present if the *IsGhosted* parameter has a nonzero (TRUE) value.

For more details about the *Servers* parameter, see RopPublicFolderIsGhosted in [MS-OXCSTOR] section 2.2.1.7.2.4.

2.2.3 RopDeleteFolder

The RopDeleteFolder operation removes a subfolder. By default, RopDeleteFolder operates only on empty folders, but it can be used on non-empty folders by setting the **DeleteFolderFlags** to also delete the subfolders and messages inside the folder.

The complete syntax of the RopDeleteFolder request and response buffers is specified in [MS-OXCROPS] section 2.2.4.3. This section specifies the syntax and semantics of various fields that are not fully specified in [MS-OXCROPS].

2.2.3.1 Request Parameter Overview

2.2.3.1.1 InputHandleIndex

The *InputHandleIndex* parameter specifies the location in the Server object handle table where the handle for the input handle is stored. The input handle for this operation is a Folder object handle.

2.2.3.1.2 DeleteFolderFlags

The *DeleteFolderFlags* parameter contains a bitmask of flags that control the folder deletion operation.

By default, RopDeleteFolder operates only on empty folders, but it can be used successfully on non-empty folders by setting two flags: DEL_FOLDERS and DEL_MESSAGES. The DEL_FOLDERS flag enables all the folder's subfolders to be removed; the DEL_MESSAGES flag enables all the folder's messages to be removed. RopDeleteFolder causes a **hard delete** of the folder if the DELETE_HARD_DELETE flag is set.

The following table lists the valid values for this flag.

Name	Value	Description
DEL_MESSAGES	0x01	If this bit is set, then delete all the messages in the folder.
DEL_FOLDERS	0x04	If this bit is set, then delete the subfolder and all its subfolders.
DELETE_HARD_DELETE	0x10	If this bit is set, then the folder is hard deleted. If it is not set, the folder is soft deleted.

If the flag DEL_MESSAGES is not used, and there are messages in the folder, neither the folder nor any of its messages will be deleted. The *ReturnValue* parameter in the response message will be "0x00000000" and the *PartialCompletion* parameter will be set to a nonzero (TRUE) value.

All other bits, if set, MUST be ignored by the server.

2.2.3.1.3 FolderId

The *FolderId* parameter contains the FID of the folder to be deleted.

2.2.3.2 Response Parameter Overview

2.2.3.2.1 ReturnValue

The *ReturnValue* parameter indicates the result of the operation. The server returns "0x00000000" to indicate success. For more details about common error codes, see [MS-OXCDATA] section 2.4.

2.2.3.2.2 PartialCompletion

If the operation fails for a subset of targets, the *PartialCompletion* parameter value is nonzero (TRUE). Otherwise, the *PartialCompletion* parameter value is 0x00 (FALSE).

2.2.4 RopSetSearchCriteria

The RopSetSearchCriteria operation establishes **search criteria** for a search folder. The search criteria are made up of a restriction (the filter to be applied) and a search scope (actual folders where the content will be searched).

A search folder contains links to the messages that meet the search criteria. The actual messages are still stored in their original locations.

The complete syntax of the RopSetSearchCriteria request and response buffers is specified in [MS-OXCROPS] section 2.2.4.4. This section specifies the syntax and semantics of various fields that are not fully specified in [MS-OXCROPS].

2.2.4.1 Request Parameter Overview

2.2.4.1.1 InputHandleIndex

The *InputHandleIndex* parameter specifies the location in the Server object handle table where the handle for the input handle is stored. The input handle for this operation is a Folder object handle.

2.2.4.1.2 RestrictionDataSize

The *RestrictionDataSize* parameter value contains the length of the *RestrictionData* parameter. If the value 0x0000 (FALSE) is passed in the *RestrictionDataSize* parameter, the search criteria that

was used most recently for this **search folder container** is used again. The value 0x0000 (FALSE) MUST NOT be passed in *RestrictionDataSize* for the search folder container's first search.

2.2.4.1.3 RestrictionData

The *RestrictionData* parameter contains a restriction. For more details about the structure of a restriction, see [MS-OXCDATA] section 2.12.

2.2.4.1.4 FolderIdCount

The *FolderIdCount* parameter contains the number of folders in the *FolderIds* parameter. If the value 0x0000 (FALSE) is passed in the *FolderIdCount* parameter, the **EntryIDs** that were used most recently to search this search folder container are used for the new search. The value 0x0000 (FALSE) MUST NOT be passed in *FolderIdCount* for the first search within a search folder container.

2.2.4.1.5 FolderIds

The *FolderIds* parameter contains a list of FIDs of the folders that will be used in the search.

2.2.4.1.6 SearchFlags

The *SearchFlags* parameter contains a bitmask of flags that control the search for a search folder.

For more details about how the *SearchFlags* parameter affects the search, see section 3.2.5.4.

The following table lists the valid values of the *SearchFlags* parameter.

Name	Value	Description
STOP_SEARCH	0x00000001	Request server to abort the search. This flag cannot be set at the same time as the RESTART_SEARCH flag.
RESTART_SEARCH	0x00000002	The search is initiated if this is the first call to RopSetSearchCriteria, or if the search is restarted, or if the search is inactive. This flag cannot be set at the same time as the STOP_SEARCH flag.
RECURSIVE_SEARCH	0x00000004	The search includes the search folder containers that are specified in the folder list in the request buffer and all their child folders. This flag cannot be set at the same time as the SHALLOW_SEARCH flag.
SHALLOW_SEARCH	0x00000008	The search only looks in the search folder containers specified in the *FolderIdList* parameter for matching entries. This flag cannot be set at the same time as the RECURSIVE_SEARCH flag. Passing neither RECURSIVE_SEARCH nor SHALLOW_SEARCH indicates that the search will use the flag from the previous execution of RopSetSearchCriteria. Also, passing neither RECURSIVE_SEARCH nor SHALLOW_SEARCH for the first search defaults to RECURSIVE_SHALLOW.
FOREGROUND_SEARCH	0x00000010	Request the server to run this search at a high priority relative to other searches. This flag cannot be set at the same time as the

Name	Value	Description
		BACKGROUND_SEARCH flag.
BACKGROUND_SEARCH	0x00000020	Request the server to run this search at normal priority relative to other searches. This flag cannot be set at the same time as the FOREGROUND_SEARCH flag. Passing neither FOREGROUND_SEARCH nor BACKGROUND_SEARCH indicates that the search will use the flag from the previous execution of RopSetSearchCriteria. Passing neither FOREGROUND_SEARCH nor BACKGROUND_SEARCH on the first search defaults to BACKGROUND_SEARCH.
CONTENT_INDEXED_SEARCH	0x00010000	Use content-indexed search exclusively.
NON_CONTENT_INDEXED_SEARCH	0x00020000	Never use content-indexed search.
STATIC_SEARCH	0x00040000	Make the search static.

All other bits SHOULD NOT be set by the client. If they are set, then the server SHOULD fail the operation with a **ResultValue** of InvalidParameter.<4>

2.2.4.2 Response Parameter Overview

2.2.4.2.1 ReturnValue

The *ReturnValue* parameter indicates the result of the operation. The value of this parameter can be one of the common error codes specified in [MS-OXCDATA] section 2.4, or one of the error codes in the following table.

Name	Value	Meaning
ecNone	0x00000000	Success
ecSearchFolderScopeViolation	0x00000490	If a search folder is searching another search folder, then it can only scope that one folder and cannot do so recursively. <5>

2.2.5 RopGetSearchCriteria

The RopGetSearchCriteria operation is used to obtain the search criteria and the status of a search for a search folder. Search criteria are created by calling RopSetSearchCriteria.

The complete syntax of the RopGetSearchCriteria request and response buffers is specified in [MS-OXCROPS] section 2.2.4.5. This section specifies the syntax and semantics of various fields that are not fully specified in [MS-OXCROPS].

2.2.5.1 Request Parameter Overview

All request parameters for this ROP are specified in [MS-OXCROPS].

2.2.5.1.1 InputHandleIndex

The *InputHandleIndex* parameter specifies the location in the Server object handle table where the handle for the input handle is stored. The input handle for this operation is a Folder object handle.

2.2.5.1.2 UseUnicode

If results are required in Unicode format, the *UseUnicode* parameter is set to a nonzero (TRUE) value; otherwise, it is set to 0x00 (FALSE).

2.2.5.1.3 IncludeRestriction

If the restriction data is required in the response, the *IncludeRestriction* parameter is set to a nonzero (TRUE) value. Otherwise, it is set to 0x00 (FALSE).

2.2.5.1.4 IncludeFolders

If the folders list is required in the response, the *IncludeFolders* parameter is set to a nonzero (TRUE) value. Otherwise, it is set to 0x00 (FALSE).

2.2.5.2 Response Parameter Overview

2.2.5.2.1 ReturnValue

The *ReturnValue* parameter indicates the result of the operation. The server returns "0x00000000" to indicate success. For more details about common error codes, see [MS-OXCDATA] section 2.4.

2.2.5.2.2 RestrictionDataSize

The *RestrictionDataSize* parameter contains the length of the *RestrictionData* parameter in bytes. If the *IncludeRestriction* parameter in the request buffer was set to 0x00 (FALSE), *RestrictionDataSize* will return 0x0000 (FALSE), regardless of the actual restriction size.

2.2.5.2.3 RestrictionData

The *RestrictionData* parameter contains a restriction that specifies the restriction for the search folder.

For more details about the structure of a restriction, see [MS-OXCDATA] section 2.13.

2.2.5.2.4 FolderIdCount

The *FolderIdCount* parameter contains the number of folders used in the search. If the *IncludeFolders* parameter in the request buffer was set to 0x0000 (FALSE), the *FolderIdCount* parameter will return 0x0000 (FALSE), regardless of the actual folder list.

2.2.5.2.5 FolderIds

The *FolderIds* parameter contains the list of FIDs of the folders that are being searched.

2.2.5.2.6 SearchFlags

The RopGetSearchCriteria operation returns a *SearchFlags* parameter that contains the state of the current search. For more details about how the flags are used by the server, see section 3.2.5.4.

The following table lists the valid values for the *SearchFlags* parameter.

Name	Value	Description
SEARCH_RUNNING	0x00000001	The search is running.
SEARCH_REBUILD	0x00000002	The search is in the CPU-intensive mode of its operation, trying to locate messages that match the criteria. If this flag is not set, the CPU-intensive part of the search's operation is over. This flag only has meaning if the search is active (if the SEARCH_RUNNING flag is set).
SEARCH_RECURSIVE	0x00000004	The search is looking in specified search folder containers and all their child search folder containers for matching entries. If this flag is not set, only the search folder containers that are explicitly included in the last call to the RopSetSearchCriteria are being searched.
SEARCH_FOREGROUND	0x00000008	The search is running at a high priority relative to other searches. If this flag is not set, the search is running at a normal priority relative to other searches.
SEARCH_COMPLETE	0x00001000	The search results are complete.
SEARCH_PARTIAL	0x00002000	The search results are not complete, as only some parts of messages were included.
SEARCH_STATIC	0x00010000	The search is static.
SEARCH_MAYBE_STATIC	0x00020000	The search is still being evaluated.
CI_TOTALLY	0x01000000	The search is completely done using content indexing.
CI_WITH_TWIR_RESIDUAL	0x02000000	The search is mostly done using content indexing.
TWIR_MOSTLY	0x04000000	The search is mostly done using store-only search.
TWIR_TOTALLY	0x08000000	The search is completely done using store-only search.

All other flags MUST be ignored by the server.

2.2.6 RopMoveCopyMessages

The RopMoveCopyMessages operation moves or copies messages from a source folder to a destination folder. This ROP applies to both public folders and private mailboxes.

If the call is being processed asynchronously, the server can return a RopProgress<6> response to indicate that the operation is still processing, or it can return a **RopMoveCopyMessages** response ([MS-OXCROPS] section 2.2.4.6) to indicate that the operation has already completed. For more details about RopProgress, including usages, restrictions, and notes, see [MS-OXCPRPT] section 2.2.22.

The complete syntax of the RopMoveCopyMessages request and response buffers is specified in [MS-OXCROPS] section 2.2.4.6. This section specifies the syntax and semantics of various fields that are not fully specified in [MS-OXCROPS].

2.2.6.1 Request Parameter Overview

2.2.6.1.1 SourceHandleIndex

The *SourceHandleIndex* parameter specifies the location in the Server object handle table where the handle for the source handle is stored. The source handle for this operation is a Folder object handle.

2.2.6.1.2 DestHandleIndex

The *DestHandleIndex* parameter specifies the location in the Server object handle table where the handle for the destination handle is stored. The destination handle for this operation is a Folder object handle.

2.2.6.1.3 MessageIdCount

The *MessageIdCount* parameter contains the number of messages to move or copy.

2.2.6.1.4 MessageIds

The *MessageIds* parameter contains a list of MIDs to move or copy. The MID structure is defined in [MS-OXCDATA] section 2.2.1.2.

2.2.6.1.5 WantAsynchronous

The *WantAsynchronous* parameter value is set to nonzero (TRUE) to request the operation of this ROP to be performed asynchronously. For the operation to be performed synchronously, the *WantAsynchronous* parameter value is set to 0x00 (FALSE). If the *WantAsynchronous* parameter value is nonzero (TRUE), the server can return a RopProgress response to indicate that the operation is still processing, or it can return a RopMoveCopyMessages response to indicate that the operation has already completed. For more details about RopProgress, including usages, restrictions and notes, see [MS-OXCPRPT] section 2.2.22.

2.2.6.1.6 WantCopy

The *WantCopy* parameter is nonzero (TRUE) if this is a copy operation, or 0x00 (FALSE) if this is a move operation.

2.2.6.2 Response Parameter Overview

2.2.6.2.1 ReturnValue

The *ReturnValue* parameter indicates the result of the operation. The server returns "0x00000000" to indicate success. For more details about common error codes, see [MS-OXCDATA] section 2.4.

2.2.6.2.2 PartialCompletion

If the operation failed for a subset of targets, the *PartialCompletion* parameter value is nonzero (TRUE). Otherwise, the *PartialCompletion* parameter value is 0x00 (FALSE).

2.2.6.3 Null Destination Failure Response Parameter Overview

2.2.6.3.1 ReturnValue

The *ReturnValue* is an unsigned 32-bit integer that specifies the status of the ROP. For this response, this field is set to 0x00000503.

2.2.6.3.2 PartialCompletion

The *PartialCompletion* parameter is an 8-bit Boolean value that specifies whether the operation failed for a subset of targets. If it did fail for a subset of targets, the value of this parameter is nonzero (TRUE). Otherwise, it is 0x00 (FALSE).

2.2.7 RopMoveFolder

The RopMoveFolder operation moves a folder from one parent to another. All the content and subfolders of the folder are moved with it.

The move can either be within a private mailbox or public folder, or between a public folder and a private mailbox.

If the call is being processed asynchronously, the server can return a RopProgress response to indicate that the operation is still processing, or it can return a RopMoveFolder response to indicate that the operation has already completed. For more details about RopProgress, including usages, restrictions, and notes, see [MS-OXCPRPT] section 2.2.22.

The complete syntax of the RopMoveFolder request and response buffers is specified in [MS-OXCROPS] section 2.2.4.7. This section specifies the syntax and semantics of various fields that are not fully specified in [MS-OXCROPS].

2.2.7.1 Request Parameter Overview

2.2.7.1.1 SourceHandleIndex

The *SourceHandleIndex* parameter specifies the location in the Server object handle table, where the handle for the source handle is stored. The source handle for this operation is a Folder object handle.

2.2.7.1.2 DestHandleIndex

The *DestHandleIndex* parameter specifies the location in the Server object handle table, where the handle for the destination handle is stored. The destination handle for this operation is a Folder object handle.

2.2.7.1.3 WantAsynchronous

The *WantAsynchronous* parameter value is set to nonzero (TRUE) to request the operation of this ROP to be performed asynchronously. For the operation to be performed synchronously, the *WantAsynchronous* parameter value is set to 0x00 (FALSE). If the *WantAsynchronous* parameter value is nonzero (TRUE), the server can return the RopProgress response to indicate that the operation is still processing, or it can return the RopMoveFolder response to indicate that the operation has already completed. For more details about RopProgress, including usages, restrictions, and notes, see [MS-OXCPRPT] section 2.2.22.

2.2.7.1.4 UseUnicode

If the *NewFolderName* parameter is formatted in Unicode, the *UseUnicode* parameter value is nonzero (TRUE); otherwise, it is set to 0x00 (FALSE).

2.2.7.1.5 FolderId

The *FolderId* parameter contains the FID of the folder to be moved.

2.2.7.1.6 NewFolderName

The *NewFolderName* parameter contains a null-terminated new folder name for the moved folder.

2.2.7.2 Response Parameter Overview

2.2.7.2.1 ReturnValue

The *ReturnValue* parameter indicates the result of the operation. The server returns "0x00000000" to indicate success. For more details about common error codes, see [MS-OXCDATA] section 2.4.

2.2.7.2.2 PartialCompletion

If the operation failed for a subset of targets, the *PartialCompletion* parameter value is nonzero (TRUE). Otherwise, the *PartialCompletion* parameter value is 0x00 (FALSE).

2.2.7.3 Null Destination Failure Response Parameter Overview

2.2.7.3.1 ReturnValue

The *ReturnValue* is an unsigned 32-bit integer that specifies the status of the ROP. For this response, this field is set to 0x00000503.

2.2.7.3.2 PartialCompletion

The *PartialCompletion* parameter is an 8-bit Boolean value that specifies whether the operation failed for a subset of targets. If it did fail for a subset of targets, the value of this parameter is nonzero (TRUE). Otherwise, it is 0x00 (FALSE).

2.2.8 RopCopyFolder

The RopCopyFolder operation creates a new folder on the destination parent folder, copying the properties and content of the source folder to the new folder. The operation can be performed between public folders and private mailboxes. All messages in the source folder are duplicated on the new folder.

If the call is being processed asynchronously, the server can return a RopProgress response to indicate that the operation is still being processed, or it can return a RopCopyFolder response to indicate that the operation has already completed. For more details about RopProgress, including usages, restrictions, and notes, see [MS-OXCPRPT] section 2.2.22.

The complete syntax of the RopCopyFolder request and response buffers is specified in [MS-OXCROPS] section 2.2.4.8. This section specifies the syntax and semantics of various fields that are not fully specified in [MS-OXCROPS].

2.2.8.1 Request Parameter Overview

2.2.8.1.1 SourceHandleIndex

The *SourceHandleIndex* parameter specifies the location in the Server object handle table where the handle for the source handle is stored. The source handle for this operation is a Folder object handle.

2.2.8.1.2 DestHandleIndex

The *DestHandleIndex* parameter specifies the location in the Server object handle table where the handle for the destination handle is stored. The destination handle for this operation is a Folder object handle.

2.2.8.1.3 WantAsynchronous

The *WantAsynchronous* parameter value is set to nonzero (TRUE) to request the operation of this ROP to be performed asynchronously. For the operation to be performed synchronously, the *WantAsynchronous* parameter value is set to 0x00 (FALSE). If the *WantAsynchronous* parameter value is nonzero (TRUE), the server can return a RopProgress response to indicate that the operation is still being processed, or it can return a RopCopyFolder response to indicate that the operation has already completed. For more details about RopProgress, including usages, restrictions, and notes, see [MS-OXCPRPT] section 2.2.22.

2.2.8.1.4 WantRecursive

The *WantRecursive* parameter is nonzero (TRUE) for all subfolders that are contained in the source folder to be duplicated in the new folder, including their properties, messages, and subfolders (in a recursive manner). Otherwise, the field is set to 0x00 (FALSE).

2.2.8.1.5 UseUnicode

If the *NewFolderName* parameter is formatted in Unicode, the *UseUnicode* parameter MUST be nonzero (TRUE). Otherwise, it is set to 0x00 (FALSE).

2.2.8.1.6 FolderId

The *FolderId* parameter contains the FID of the folder to copy.

2.2.8.1.7 NewFolderName

The *NewFolderName* parameter contains a null-terminated new folder name string for the copied folder.

2.2.8.2 Response Parameter Overview

2.2.8.2.1 ReturnValue

The *ReturnValue* parameter indicates the result of the operation. The server returns "0x00000000" to indicate success. For more details about common error codes, see [MS-OXCDATA] section 2.4.

2.2.8.2.2 PartialCompletion

If the operation failed for a subset of targets, the *PartialCompletion* parameter value is nonzero (TRUE). Otherwise, the *PartialCompletion* parameter value is 0x00 (FALSE).

2.2.8.3 Null Destination Failure Response Parameter Overview

2.2.8.3.1 ReturnValue

The *ReturnValue* is an unsigned 32-bit integer that specifies the status of the ROP. For this response, this field is set to 0x00000503.

2.2.8.3.2 PartialCompletion

The *PartialCompletion* parameter is an 8-bit Boolean value that specifies whether the operation failed for a subset of targets. If it did fail for a subset of targets, the value of this parameter is nonzero (TRUE). Otherwise, it is 0x00 (FALSE).

2.2.9 RopEmptyFolder

The RopEmptyFolder operation is used to soft delete all messages and subfolders from a folder without deleting the folder itself. To hard delete all messages and subfolders from a folder, use RopHardDeleteMessagesAndSubfolders.

Messages that do not exist, have been moved elsewhere, are open with read/write access, or are currently submitted MUST NOT be deleted by the server.

If the call is being processed asynchronously, the server can return a RopProgress response to indicate that the operation is still processing, or it can return a RopEmptyFolder response to indicate that the operation has already completed. For more details about RopProgress, including usages, restrictions, and notes, see [MS-OXCPRPT] section 2.2.22.

The complete syntax of the RopEmptyFolder request and response buffers is specified in [MS-OXCROPS] section 2.2.4.9. This section specifies the syntax and semantics of various fields that are not fully specified in [MS-OXCROPS].

2.2.9.1 Request Parameter Overview

2.2.9.1.1 InputHandleIndex

The *InputHandleIndex* parameter specifies the location in the Server object handle table where the handle for the input handle is stored. The input handle for this operation is a Folder object handle.

2.2.9.1.2 WantAsynchronous

The *WantAsynchronous* parameter value is set to nonzero (TRUE) to request the operation of this ROP to be performed asynchronously. For the operation to be performed synchronously, the *WantAsynchronous* parameter value is set to 0x00 (FALSE). If the *WantAsynchronous* parameter value is nonzero (TRUE), the server can return a RopProgress response to indicate that the operation is still processing, or it can return a RopEmptyFolder response to indicate that the operation has already completed. For more details about RopProgress, including usages, restrictions, and notes, see [MS-OXCPRPT] section 2.2.22.

2.2.9.1.3 WantDeleteAssociated

To delete all messages, including the folder associated information (FAI) messages, the *WantDeleteAssociated* parameter value MUST be nonzero (TRUE); otherwise, the *WantDeleteAssociated* parameter value is set to 0x00 (FALSE).

2.2.9.2 Response Parameter Overview

2.2.9.2.1 ReturnValue

The *ReturnValue* parameter indicates the result of the operation. The server returns "0x00000000" to indicate success. For more details about common error codes, see [MS-OXCDATA] section 2.4.

2.2.9.2.2 PartialCompletion

If the operation failed for a subset of targets, the *PartialCompletion* parameter value is nonzero (TRUE). Otherwise, the *PartialCompletion* parameter value is 0x00 (FALSE).

2.2.10 RopHardDeleteMessagesAndSubfolders

The RopHardDeleteMessagesAndSubfolders operation is used to hard delete all messages and subfolders from a folder without deleting the folder itself.

Messages that do not exist, have been moved elsewhere, are open with read/write access, or are currently submitted MUST NOT be deleted.

If the call is being processed asynchronously, the server can return a RopProgress response to indicate that the operation is still processing, or it can return a RopHardDeleteMessagesAndSubfolders response to indicate that the operation has already completed. For more details about RopProgress, including usages, restrictions, and notes, see [MS-OXCPRPT] section 2.2.22.

The complete syntax of the RopHardDeleteMessagesAndSubfolders request and response buffers are specified in [MS-OXCROPS] section 2.2.4.10. This section specifies the syntax and semantics of various fields that are not fully specified in [MS-OXCROPS].

2.2.10.1 Request Parameter Overview

2.2.10.1.1 InputHandleIndex

The *InputHandleIndex* parameter specifies the location in the Server object handle table where the handle for the input handle is stored. The input handle for this operation is a Folder object handle.

2.2.10.1.2 WantAsynchronous

The *WantAsynchronous* parameter value is set to nonzero (TRUE) to request that the operation of this ROP be performed asynchronously. For the operation to be performed synchronously, the *WantAsynchronous* parameter value is set to 0x00 (FALSE). If the *WantAsynchronous* parameter value is nonzero (TRUE), the server can return a RopProgress response to indicate that the operation is still processing, or it can return a RopHardDeleteMessagesAndSubfolders response to indicate that the operation has already completed. For more details about RopProgress, including usages, restrictions, and notes, see [MS-OXCPRPT] section 2.2.22.

2.2.10.1.3 WantDeleteAssociated

To delete all messages, including the FAI messages, the *WantDeleteAssociated* parameter value MUST be nonzero (TRUE). Otherwise, the *WantDeleteAssociated* parameter value is set to 0x00 (FALSE).

2.2.10.2 Response Parameter Overview

2.2.10.2.1 ReturnValue

The *ReturnValue* parameter indicates the result of the operation. The server returns "0x00000000" to indicate success. For more details about common error codes, see [MS-OXCDATA] section 2.4.

2.2.10.2.2 PartialCompletion

If the operation failed for a subset of targets, the *PartialCompletion* parameter value is nonzero (TRUE). Otherwise, the *PartialCompletion* parameter value is 0x00 (FALSE).

2.2.11 RopDeleteMessages

The RopDeleteMessages operation deletes one or more messages from a folder. Messages that do not exist, have been moved elsewhere, are open with read/write access, or are currently submitted MUST NOT be deleted. Messages deleted with this ROP are soft deleted.

If the call is being processed asynchronously, the server can return a RopProgress response to indicate that the operation is still processing, or it can return a RopDeleteMessages response to indicate that the operation has already completed. For more details about RopProgress, including usages, restrictions, and notes, see [MS-OXCPRPT] section 2.2.22.

The complete syntax of the RopDeleteMessages request and response buffers is specified in [MS-OXCROPS] section 2.2.4.11. This section specifies the syntax and semantics of various fields that are not fully specified in [MS-OXCROPS].

2.2.11.1 Request Parameter Overview

2.2.11.1.1 InputHandleIndex

The *InputHandleIndex* parameter specifies the location in the Server object handle table where the handle for the input handle is stored. The input handle for this operation is a Folder object handle.

2.2.11.1.2 WantAsynchronous

The *WantAsynchronous* parameter value is set to nonzero (TRUE) to request the operation of this ROP to be performed asynchronously. For the operation to be performed synchronously, the *WantAsynchronous* parameter value is set to 0x00 (FALSE). If the *WantAsynchronous* parameter value is nonzero (TRUE), the server can return a RopProgress response to indicate that the operation is still processing, or it can return a RopDeleteMessages response to indicate that the operation has already completed. For more details about RopProgress, including usages, restrictions, and notes, see [MS-OXCPRPT] section 2.2.22.

2.2.11.1.3 NotifyNonRead

If the *NotifyNonRead* parameter is 0x00 (FALSE), the server does not generate a non-read receipt for the deleted messages. If the *NotifyNonRead* parameter value is nonzero (TRUE), the server

generates non-read receipts for the messages that are being deleted and have requested **read receipts**. A non-read receipt is a notice that a message was deleted before it was read. For more information about read receipts, see [MS-OXOMSG] section 1.3.2.

2.2.11.1.4 MessageIdCount

The *MessageIdCount* parameter contains the number of messages to delete.

2.2.11.1.5 MessageIds

The *MessageIds* parameter contains the list of MIDs of the messages to be deleted.

2.2.11.2 Response Parameter Overview

2.2.11.2.1 ReturnValue

The *ReturnValue* parameter indicates the result of the operation. The server returns "0x00000000" to indicate success. For more details about common error codes, see [MS-OXCDATA] section 2.4.

2.2.11.2.2 PartialCompletion

If the operation failed for a subset of targets, the *PartialCompletion* parameter value is nonzero (TRUE). Otherwise, the *PartialCompletion* parameter value is 0x00 (FALSE).

2.2.12 RopHardDeleteMessages

The RopHardDeleteMessages operation hard deletes one or more messages that are listed in the request buffer. Messages that do not exist, have been moved elsewhere, are opened with read/write access, or are currently submitted MUST NOT be deleted.

If the call is being processed asynchronously, the server can return a RopProgress response to indicate that the operation is still processing, or it can return a RopHardDeleteMessages response to indicate that the operation has already completed. For more details about RopProgress, including usages, restrictions, and notes, see [MS-OXCPRPT] section 2.2.22.

The complete syntax of the RopHardDeleteMessages request and response buffers is specified in [MS-OXCROPS] section 2.2.4.12. This section specifies the syntax and semantics of various fields that are not fully specified in [MS-OXCROPS].

2.2.12.1 Request Parameter Overview

2.2.12.1.1 InputHandleIndex

The *InputHandleIndex* parameter specifies the location in the Server object handle table where the handle for the input handle is stored. The input handle for this operation is a Folder object handle.

2.2.12.1.2 WantAsynchronous

The *WantAsynchronous* parameter value is set to nonzero (TRUE) to request that the operation of this ROP be performed asynchronously. For the operation to be performed synchronously, the *WantAsynchronous* parameter value is set to 0x00 (FALSE). If the *WantAsynchronous* parameter value is nonzero (TRUE), the server can return a RopProgress response to indicate that the operation is still processing, or it can return a RopHardDeleteMessages response to indicate that the

operation has already completed. For more details about RopProgress, including usages, restrictions, and notes, see [MS-OXCPRPT] section 2.2.22.

2.2.12.1.3 NotifyNonRead

If the *NotifyNonRead* parameter is set to 0x00 (FALSE), the server does not generate a non-read receipt for the deleted messages. If the *NotifyNonRead* parameter value is set to nonzero (TRUE), the server generates non-read receipts for the messages that are being deleted and have requested read receipts. A non-read receipt is a notice that a message was deleted before it was read. For more information about read receipts, see [MS-OXOMSG] section 1.3.2.

2.2.12.1.4 MessageIdCount

The *MessageIdCount* parameter contains the number of messages to delete.

2.2.12.1.5 MessageIds

The *MessageIds* parameter contains the list of MIDs of the messages to be deleted.

2.2.12.2 Response Parameter Overview

2.2.12.2.1 ReturnValue

The *ReturnValue* parameter indicates the result of the operation. The server returns "0x00000000" to indicate success. For more details about common error codes, see [MS-OXCDATA] section 2.4.

2.2.12.2.2 PartialCompletion

If the operation failed for a subset of targets, the *PartialCompletion* parameter value is nonzero (TRUE). Otherwise, the *PartialCompletion* parameter value is 0x00 (FALSE).

2.2.13 RopGetHierarchyTable

The RopGetHierarchyTable operation is used to retrieve the **hierarchy table** for a folder. This ROP returns a **Table object** on which table operations can be performed. For more details about Table objects and table operations, see [MS-OXCTABL].

The client application MUST send a RopRelease request after executing all subsequent operations on the table handle obtained by using this ROP.

The complete syntax of the RopGetHierarchyTable request and response buffers is specified in [MS-OXCROPS] section 2.2.4.13. This section specifies the syntax and semantics of various fields that are not fully specified in [MS-OXCROPS].

2.2.13.1 Request Parameter Overview

2.2.13.1.1 InputHandleIndex

The *InputHandleIndex* parameter specifies the location in the Server object handle table where the handle for the input handle is stored. The input handle for this operation is a Folder object handle.

2.2.13.1.2 OutputHandleIndex

The *OutputHandleIndex* parameter specifies the location in the Server object handle table where the handle for the output handle is stored. The output handle for this operation is a Table object handle. For more details about Table objects, see [MS-OXCTABL].

2.2.13.1.3 TableFlags

The *TableFlags* parameter contains a bitmask of flags that control how information is returned in the table.

The following table lists valid values of the *TableFlags* parameter.

Name	Bitmask	Description
Depth	0x04	Fills the hierarchy table with search folder containers from all levels. If this flag is not set, the hierarchy table contains only the search folder container's immediate child search folder containers.
DeferredErrors	0x08	The ROP response can return immediately, possibly before the ROP execution is complete, and in this case, the *ReturnValue* as well the **RowCount** fields in the return buffer might not be accurate. Only *ReturnValues* reporting failure can be considered valid in this case.
NoNotifications	0x10	Disables all notifications on this Table object.
SoftDeletes	0x20	Enables the client to get a list of the soft-deleted folders.
UseUnicode	0x40	Requests that the columns that contain string data be returned in Unicode format. If *UseUnicode* is not present, then the string data will be encoded in the codepage of the logon.
SuppressesNotifications	0x80	Suppresses notifications generated by this client's actions on this Table object.

Clients SHOULD NOT set any flag values other than those defined in this table. If a client sets an undefined flag value, then servers SHOULD fail the operation with an error code of InvalidParameter.<7>

2.2.13.2 Response Parameter Overview

2.2.13.2.1 ReturnValue

The *ReturnValue* parameter indicates the result of the operation. The server returns "0x00000000" to indicate success. For more details about common error codes, see [MS-OXCDATA] section 2.4.

2.2.13.2.2 RowCount

The *RowCount* parameter contains the number of rows in the hierarchy table. This field can be 0x00000000 (FALSE) instead of the actual count if the DeferredErrors flag is used.

2.2.14 RopGetContentsTable

The RopGetContentsTable operation is used to retrieve the **contents table** for a folder. This ROP returns a Table object on which table operations can be performed. For more details about Table objects and table operations, see [MS-OXCTABL].

The client application MUST send a RopRelease request after executing all subsequent operations on the table handle obtained by using this ROP.

The complete syntax of the RopGetContentsTable request and response buffers is specified in [MS-OXCROPS] section 2.2.4.14. This section specifies the syntax and semantics of various fields that are not fully specified in [MS-OXCROPS].

2.2.14.1 Request Parameter Overview

2.2.14.1.1 InputHandleIndex

The *InputHandleIndex* parameter specifies the location in the Server object handle table where the handle for the input handle is stored. The input handle for this operation is a Folder object handle.

2.2.14.1.2 OutputHandleIndex

The *OutputHandleIndex* parameter specifies the location in the Server object handle table where the handle for the output handle is stored. The output handle for this operation is a Table object handle. For more details about Table objects, see [MS-OXCTABL].

2.2.14.1.3 TableFlags

The *TableFlags* parameter contains a bitmask of flags that control how information is returned in the table.

The following table lists the valid values of the *TableFlags* parameter.

Name	Bitmask	Description
None	0x00	No flags enabled.
Associated<8>	0x02	Requests an FAI table instead of a standard table. For more information about FAI messages, see [MS-OXCMSG] section 1.3.2.
DeferredErrors	0x08	The call can return immediately, possibly before the ROP execution is complete and in this case the **ReturnValue** and the **RowCount** fields in the return buffer might not be accurate. Only **ReturnValues** reporting failure can be considered valid in this case.
NoNotifications	0x10	Disables table notifications to the client.
SoftDeletes	0x20	Enables the client to get a list of the soft deleted messages in a folder and to either restore the messages back to the original folders or permanently remove the messages from the system.
UseUnicode<9>	0x40	Requests that the columns that contain string data of unspecified type be returned in Unicode format. If UseUnicode is not present, then the string data is encoded in the codepage of the Logon.

Clients SHOULD NOT set any bits other than those listed in this table. If a client sets a bit not listed in this table, then the server SHOULD fail the operation with an error code of InvalidOperation.<10>

2.2.14.2 Response Parameter Overview

2.2.14.2.1 ReturnValue

The *ReturnValue* parameter indicates the result of the operation. The server returns "0x00000000" to indicate success. For more details about common error codes, see [MS-OXCDATA] section 2.4.

2.2.14.2.2 RowCount

The *RowCount* parameter contains the number of rows in the table. This field can be 0x00000000 (FALSE) instead of the actual count. It is at the server's discretion as to whether the DeferredErrors flag is used.

2.3 Folder Object Properties

Folder objects can be created and modified by clients and servers. Except where noted, this section defines constraints to which both clients and servers adhere when operating on Folder objects.

Unless otherwise specified, a Folder object adheres to all property constraints specified in [MS-OXPROPS]. A Folder object can also contain other properties, as specified in [MS-OXOSFLD], [MS-OXOSRCH], and [MS-OXPROPS].

When a property is referred to as read-only, it means that clients SHOULD NOT try to change the value of this property and servers return an error and ignore any request to change the value of the property.

2.3.1 General Properties

The following properties exist on Folder objects as well as on other messaging objects. These properties are set by the server and are read-only to the client. For details about the following properties, see [MS-OXCPRPT] section 2.2.1.

PidTagAccess

PidTagAccessLevel

PidTagChangeKey

PidTagCreationTime

PidTagLastModificationTime

PidTagLastModifierName

PidTagObjectType

PidTagRecordKey

PidTagSearchKey

2.3.2 Folder Object Specific Properties

The following properties are available on Folder objects.

2.3.2.1 Read-Only Properties

2.3.2.1.1 PidTagContentCount

A **PtypInteger32** property that specifies the number of messages in a folder, as computed by the message store. The value does not include folder associated information (FAI) entries in the folder. For more information about FAI and non-FAI messages, see [MS-OXCMSG] section 1.3.2.

2.3.2.1.2 PidTagContentUnreadCount

A **PtypInteger32** property that specifies the number of unread messages in a folder, as computed by the message store.

2.3.2.1.3 PidTagDeletedOn

A **PtypTime** property that specifies the time when the item or folder was soft deleted.

2.3.2.1.4 PidTagAddressBookEntryId

A **PtypBinary** property that contains the name-service EntryID of a directory object that refers to a public folder. This property is only set for public folders. For more information about public folders, see [MS-OXCSTOR] section 1.3.1.

2.3.2.1.5 PidTagFolderId

A **PtypInteger64** property that contains the FID of the folder.

2.3.2.1.6 PidTagHierarchyChangeNumber

A **PtypInteger32** property that monotonically increases every time a subfolder is added to or deleted from this folder.

2.3.2.1.7 PidTagMessageSize

A **PtypInteger32** property that contains the aggregate size of messages in the folder.

2.3.2.1.8 PidTagMessageSizeExtended

A **PtypInteger64** property that specifies the 64-bit version of the PidTagMessageSize property.

2.3.2.1.9 PidTagSubfolders

A **PtypBoolean** property that specifies whether this folder has any subfolders.

2.3.2.2 Read/Write Properties

2.3.2.2.1 PidTagAttributeHidden

A **PtypBoolean** property that specifies the hide or show status of a folder. The folder SHOULD be hidden by the client if the PidTagAttributeHidden property is nonzero; otherwise, the folder SHOULD NOT be hidden.

2.3.2.2.2 PidTagComment

A **PtypString** property that contains a comment about the purpose or content of the folder.

2.3.2.2.3 PidTagDisplayName

A **PtypString** property that specifies the display name of the folder.

Folders require sibling subfolders to have unique display names.

2.3.2.2.4 PidTagFolderType

A **PtypInteger32** property that specifies the type of the folder.

The following table contains the valid values of the PidTagFolderType property.

Folder type	Value	Description
FOLDER_ROOT	0x00000000	The root folder of the folder hierarchy table, that is, a folder that has no parent folder.
FOLDER_GENERIC	0x00000001	A generic folder that contains messages and other folders.
FOLDER_SEARCH	0x00000002	A folder that contains the results of a search, in the form of links to messages that meet search criteria.

2.3.2.2.5 PidTagRights

A **PtypInteger32** property that specifies the client's folder permissions. For more details about folder permissions and valid PidTagRights values, see the values provided for the PidTagMemberRights property in [MS-OXCPERM] section 2.2.1.6. Note that the **FreeBusyDetailed** and **FreeBusySimple** flags mentioned in the PidTagMemberRights property description do not apply to the PidTagRights property.

3 Protocol Details

3.1 Client Details

3.1.1 Abstract Data Model

This section describes a conceptual model of possible data organization that an implementation maintains to participate in this protocol. The described organization is provided to facilitate the explanation of how the protocol behaves. This document does not mandate that implementations adhere to this model, as long as their external behavior is consistent with that specified in this document.

3.1.1.1 Hierarchy Table

A hierarchy table contains information about the folders in a message store. Each row of a hierarchy table contains a set of columns with information about one folder. Hierarchy tables are used primarily by clients and implemented by message-store providers to show a tree of folders and subfolders.

The following are the two hierarchy tables:

- Standard
- Soft deleted

The standard table contains only folders that were not deleted. The soft deleted table contains only folders that have been soft deleted.

A hierarchy table can be accessed by using RopGetHierarchyTable (section 2.2.13).

3.1.1.2 Contents Table

A contents table contains information about objects in a message search folder container. The contents table of a folder lists information about its messages.

The following are the four folder contents tables:

- Standard
- Standard soft deleted
- Folder associated information (FAI)
- FAI soft deleted

Standard contents tables contain only standard (non-FAI) messages. FAI tables contain only FAI messages. For more information about Folder associated information (FAI) messages, see [MS-OXCMSG] section 1.3.2.

The soft deleted views contain only messages that have been soft deleted.

A contents table is obtained by using RopGetContentsTable (section 2.2.14).

3.1.2 Timers

None.

3.1.3 Initialization

None.

3.1.4 Higher-Layer Triggered Events

3.1.4.1 Open a Folder

Before any data can be read from or written to a folder, an implementation needs to ensure that the folder exists, and either opens the folder or creates it if it does not exist. Also, a client requires sufficient access rights to the folder for this operation to succeed.

To open an existing folder, an implementation sends the RopOpenFolder request. In order to send this request, the implementation first obtains the FID for the Folder object to be opened. The FID can be retrieved from the hierarchy table that contains the folder's information by including the PidTagFolderId property in a RopSetColumns request ([MS-OXCROPS] section 2.2.5.1.1). The *InputHandleIndex* that is returned by RopSetColumns can be used in subsequent operations on the opened folder. After all data manipulation on this folder is done, a RopRelease request MUST be sent.

3.1.4.2 Create a Folder

Before any data can be read from or written to a folder, an implementation needs to ensure that the folder exists, and open or create it, if it does not exist.

Before a folder can be created, the parent folder MUST already exist.

To create a folder, or open an existing folder by its name, an implementation sends the RopCreateFolder request. The parameters that are returned by this ROP can be used in subsequent operations on the created/opened folder. After all data manipulation on this folder is done, a RopRelease request MUST be sent.

3.1.4.3 Delete a Folder

To be deleted, a folder MUST exist, and the client application needs the access rights to delete it. Also, if the folder is not empty, the client application sets the *DeleteFolderFlags* parameter to delete all existing subfolders and messages. The *DeleteFolderFlags* parameter can also be used to specify a hard deletion, when the DELETE_HARD_DELETE flag is set. Besides the ReturnValue, this operation returns a PartialCompletion flag that indicates whether there are any subfolders or messages that could not be deleted, and, consequently, that the folder was not deleted.

3.1.4.4 Set Search Criteria

Clients create a search folder by calling RopCreateFolder with the *FolderType* input parameter set to a search folder type. Clients fill a search folder by setting up search criteria, or **rules**, that serve to filter out messages that have particular characteristics. Search criteria are set up by calling RopSetSearchCriteria.

To set the search criteria in a folder, the implementation builds restriction structures to represent the search criteria to be applied, and specifies FIDs of folders to be used as the search scope. Then, the implementation sends a RopSetSearchCriteria request, specifying a set of flags that control the details of how the search is performed. After that, the client sends a RopGetContentsTable request to access the search folder's contents table, and the messages that match the criteria appear in the table.

When the client is finished using a search folder, the folder can either be deleted or remain open for later use. Note that if the search folder is deleted, only message links are deleted. The actual messages remain in their parent folders.

3.1.4.5 Get Search Criteria

RopGetSearchCriteria is used to obtain the search criteria and the status of a search for a search folder. Search criteria are created by sending a RopSetSearchCriteria request.

To obtain the search criteria and search status of a search folder, the client application sends a RopGetSearchCriteria request with the appropriate flags set in the request buffer of the ROP.

3.1.4.6 Move or Copy Messages

RopMoveCopyMessages moves or copies the specified messages from the source folder to the destination folder.

The implementation sends a RopMoveCopyMessages request, which sets the flag parameters properly, identifies the operation (copy or move) and mode (synchronous or asynchronous), and also includes a list of MIDs for the messages to be either moved or copied.

3.1.4.7 Move Folder

RopMoveFolder moves a folder from one parent to another. All the properties, contents, and subfolders of the folder are moved with the folder.

The implementation sends a RopMoveFolder request, which sets the flag parameters properly, and identifies the mode (synchronous or asynchronous) and the new folder name.

3.1.4.8 Copy Folder

RopCopyFolder creates a new folder under the destination folder, and copies the properties and contents of the source folder to the new folder. All the messages in the source folder are duplicated in the new folder. If the WantRecursive flag is used, the subfolders that are contained in the source folder are also duplicated in the new folder, including their properties, messages, and subfolders (in a recursive manner).

The implementation sends a RopCopyFolder request, which sets the flag parameters properly, and identifies the mode (synchronous or asynchronous), the new folder name's locale, and the new folder name.

3.1.4.9 Empty a Folder

RopEmptyFolder and RopHardDeleteMessagesAndSubfolders are used to delete all messages and subfolders from a folder without deleting the folder itself. RopEmptyFolder is used to soft delete and RopHardDeleteMessagesAndSubfolders is used to permanently delete all messages and subfolders from a folder. Both ROPs behave in the same way and require the same request parameters.

3.1.4.10 Delete Messages

To remove existing messages from folders, the client application can use the RopDeleteMessages to have the messages soft deleted or RopHardDeleteMessages to have the messages permanently removed from the database.

3.1.4.11 Hierarchy Table

RopGetHierarchyTable returns a Table object that contains information about the folders in a message store.

To manipulate a hierarchy table object that is associated with a folder, the implementation sends a RopGetHierarchyTable request by using the appropriate table flags. Subsequent operations can be executed on the opened table and a RopRelease request on the Table object MUST be sent after all table manipulation has been done.

3.1.4.12 Contents Table

RopGetContentsTable returns a Table object that contains information about messages in a message search folder container.

To manipulate a Table object associated with a folder, the implementation sends a RopGetContentsTable request by using the appropriate table flags. Subsequent operations can be executed on the opened table and a RopRelease request on the Table object MUST be sent after all table manipulation has been completed.

3.1.5 Message Processing Events and Sequencing Rules

The following ROPs can get a RopProgress response from the server instead of their own response ROP:

RopHardDeleteMessagesAndSubfolders

RopEmptyFolder

RopHardDeleteMessages

RopDeleteMessages

RopMoveCopyMessages

RopMoveFolder

RopCopyFolder

The client can receive a RopProgress response after one of the above ROPs has been sent if the ROP request was sent with the *WantAsynchronous* parameter set to a nonzero value. In this case, the client can send RopProgress requests to abort an in-progress operation or to get information about the progress and/or the final status of the operation. For more details about RopProgress, including usages, restrictions, and notes, see [MS-OXCPRPT] section 2.2.22.

3.1.6 Timer Events

None.

3.1.7 Other Local Events

None.

3.2 Server Details

3.2.1 Abstract Data Model

This section describes a conceptual model of possible data organization that an implementation maintains to participate in this protocol. The described organization is provided to facilitate the explanation of how the protocol behaves. This document does not mandate that implementations adhere to this model, as long as their external behavior is consistent with that specified in this document.

The abstract data model used by the server and the client are the same.

3.2.2 Timers

None.

3.2.3 Initialization

None.

3.2.4 Higher-Layer Triggered Events

Various agents on the server could issue the same higher-layer triggered events, as specified in section 3.1.4. The same considerations specified in section 3.1.4 for client implementations also apply to server implementations.

3.2.5 Message Processing Events and Sequencing Rules

3.2.5.1 RopOpenFolder

RopOpenFolder provides access to an existing folder in the mailbox store. The object that is returned by this ROP can then be used on subsequent ROPs, such as RopGetPropertiesSpecific to get properties, or RopGetContentsTable to query the contents in that folder. For more information about these ROPs, see [MS-OXCROPS] section 2.2.

RopOpenFolder will succeed only if a folder with the specified ID actually exists and the client has sufficient access rights to view the folder.

If a folder was previously soft deleted, it can be accessed by using the OpenSoftDeleted flag. If this flag is used, RopOpenFolder provides access to folders that are soft deleted and to folders that are not soft deleted. If this flag is not used, RopOpenFolder only provides access to folders that are not soft deleted.

The following specific error codes apply to this ROP. For more details about ROP errors, see [MS-OXCDATA] section 2.4.

Name	Value	Meaning
ecNotFound	0x8004010F	The FID does not correspond to a folder in the database. OR The client does not have rights to the folder. OR The folder is soft deleted and the client has not specified OpenSoftDeleted.

Name	Value	Meaning
ecNotSupported	0x80040102	The object that this ROP was called on is not of type folder or Logon.

3.2.5.2 RopCreateFolder

RopCreateFolder creates a new folder in the database and provides access to it by returning a Folder object, which can be used in subsequent ROPs, similar to the one that is returned by RopOpenFolder. Unlike RopCreateMessage, RopCreateFolder immediately creates the folder on the database and does not require a call to another ROP to commit the transaction.

A folder name MUST be specified to create a folder. A folder description is optional. The folder name MUST be unique within the parent folder. In other words, sibling folders cannot have the same name.

If a folder with the same name already exists, and *OpenExisting* flag is not used, RopCreateFolder fails with error code ecDuplicateName.

If a folder with the same name already exists, and *OpenExisting* flag is used, RopCreateFolder returns the existing folder, as if RopOpenFolder was called.

If a folder with the same name does not exist, RopCreateFolder will create a new folder, regardless of the value of *OpenExisting*.

The following specific error codes apply to this ROP. For more details about ROP errors, see [MS-OXCDATA] section 2.4.

Name	Value	Meaning
ecInvalidParam	0x80070057	*FolderType* was specified as a search folder on a public folders store.
ecaccessdenied	0x80070005	The client does not have permissions to create this folder. OR The object that this ROP is called on is a soft deleted folder.
ecDuplicateName	0x80040604	A folder with the same name already exists, and the *OpenExisting* flag was not specified.
ecNotSupported	0x80040102	The object that this ROP was called on is not a Folder object.

3.2.5.3 RopDeleteFolder

RopDeleteFolder removes an existing folder from the database.

If the DELETE_HARD_DELETE flag is specified, the folder MUST be removed and can no longer be accessed by the client with subsequent ROPs. If the DELETE_HARD_DELETE flag is not specified, the folder becomes soft deleted.

The following specific error codes apply to this ROP. For more details about ROP errors, see [MS-OXCDATA] section 2.4.

Name	Value	Meaning
ecaccessdenied	0x80070005	An attempt was made to delete the Root folder. OR

Name	Value	Meaning
		An attempt was made to delete a **special folder**. OR Client does not have permissions to delete this folder.
ecNotFound	0x8004010F	Folder with the specified ID does not exist, or the client has no access to view that folder.
ecFolderHasChildren	0x80040609	The folder has subfolders and the DEL_FOLDERS flag was not specified.
ecNotSupported	0x80040102	The object that this ROP was called on is not a Folder object.

3.2.5.4 RopSetSearchCriteria

RopSetSearchCriteria modifies the search criteria of a search folder. The search criteria are made up of a restriction and a search scope (the actual folders where the content will be searched).

Clients create a search folder by calling RopCreateFolder with the *FolderType* input parameter set to search folder type. Clients fill a search folder by setting up and applying search criteria that determine which messages are included in the folder with particular characteristics. The search criteria are specified by using RopSetSearchCriteria. RopSetSearchCriteria uses restrictions created by the client and the list of folders that indicate the search scope to identify the messages that match the specified restriction. The messages that satisfy the criteria appear as links in the search folder. When the client calls RopGetContentsTable to access the search folder's contents table, the selected messages appear in the table. Contents tables for search folders contain the same columns as contents tables for generic folders. However, for search folders, the PidTagParentEntryId property is the EntryID of the folder where the linked message resides. For more details about restrictions, see [MS-OXCDATA] section 2.13. For more details about search folders, see [MS-OXOSRCH].

When the search results are retrieved, a client can choose to keep the folder for later use or to delete it. When the search folder is deleted, the **Message objects** found in the search are not deleted, and the actual messages remain in their parent folders.

After search criteria are applied to a search folder, the client can query the contents of the search folder by using **RopGetContents** in the same way that the client would query for contents of a normal folder.

When the new search criteria are applied, the search folder modifies its contents to include only the items that match the new search criteria. The ROP response can return before the contents are fully updated.

For dynamic search folders, the contents of the search folder MUST continue to be updated as messages move around the mailbox and start to match or cease to match the search criteria.

For static search folders, the contents of the search folder are not updated after the initial population is complete.

The server can use context indexing by default. This decision is at the discretion of the server implementation, and is usually based on the nature of the restriction that is used. When using context indexing in searches, the server allows the client to quickly search text in messages through the use of pre-built indexes, while non-content-indexed searches are based on a sequential scan of all the messages in the search scope.

Some differences between content-indexed and non-content-indexed searches are listed in the following table.

Content-indexed search	Non-content-indexed search
Based on words, phrases, and sentences.	Based on a **stream** of bytes.
Ignores punctuation and spaces, and is also not case sensitive.	Finds only an exact match of all characters.
Searches within attachment types that are supported by the installed filters.	Does not search within attachments.
Uses **full-text index** to locate records.	Performs a serial scan of the entire folder.
Supports only full-text searches.	Supports the full set of restrictions, which includes non-text **property types** such as date and time.

If the NON_CONTENT_INDEXED_SEARCH flag is used, the search does not use a **full-text search**.

If the NON_CONTENT_INDEXED_SEARCH flag is not used, the server uses a content-indexed search for text searches, in which case the search is static, regardless of the STATIC_SEARCH flag.

If the STATIC_SEARCH flag is used, the search is static.

If the NON_CONTENT_INDEXED_SEARCH flag is used and the STATIC_SEARCH flag is not used, the search is a dynamic search, and not a full-text search.

If the STOP_SEARCH flag is used, the server SHOULD stop the initial population of the search folder. Due to the asynchronous nature of the call, the server can complete the operation before the RopSetSearchCriteria with STOP_SEARCH is serviced. The server can take some time to stop and might not stop at all.

If the RESTART_SEARCH flag is used, the server restarts the population of the search folder.

If neither the STOP_SEARCH flag nor RESTART_SEARCH flag are used, the search continues in the previous state and either continues populating or not.

A static search causes the search folder to be populated once with all messages that match the search criteria at the point in time where the search is restarted. The search folder MUST NOT be updated with messages that enter or exit the search criteria after the initial population. To trigger an update, another RopSetSearchCriteria with RESTART_SEARCH flag is required.

A dynamic search causes the search folder to be initially populated with all messages that match the search criteria at the point in time when the search is restarted. The search folder will continue to be updated with messages that enter or exit the search criteria. Calling RopSetSearchCriteria with the STOP_SEARCH flag does not have any effect on a dynamic folder that has already completed its initial population. STOP_SEARCH does not stop the dynamic nature of the search folder.

If the client needs to know when the initial population of the search folder has been compiled, the client can issue RopGetSearchCriteria, and if the SEARCH_RUNNING flag is returned, the initial population of the search folder is still being compiled.

Another way to know when the initial population of the search folder is compiled is to use RopRegisterNotification ([MS-OXCNOTIF] section 2.2.1.2.1) on the search folder, and wait for the **SearchComplete** event.

Note the term initial population of the search folder. For a static search, this term refers to the single population of the search folder. For dynamic searches, this term refers to the population of the search folder at the moment in time when the search criterion is changed. The dynamic search folders will continue to update even after the initial population is completed.

The following specific error codes apply to this ROP. For more details about ROP errors, see [MS-OXCDATA] section 2.4.

Name	Value	Meaning
ecNotInitialized	0x80040605	No FIDs have been specified for this search folder. Note that if the FIDs were specified on a previous call to RopSetSearchCriteria and no IDs are specified in the next RopSetSearchCriteria call, the previous IDs will continue to be used. If FIDs are specified, they will override previous IDs.
ecNotSearchFolder	0x00000461	The object is not a search folder.
ecTooComplex	0x80040117	The restriction is too complex.
ecNotSupported	0x80040102	The object that this ROP was called on is not a Folder object. OR The request tried to perform a recursive search on a public folder.

3.2.5.5 RopGetSearchCriteria

RopGetSearchCriteria returns the current search criteria (only if the requesting ROP actually asked for this criteria) and the state of the search for a search folder.

The following specific error codes apply to this ROP. For more details about ROP errors, see [MS-OXCDATA] section 2.4.

Name	Value	Meaning
ecNotSearchFolder	0x00000461	The object is not a search folder.
ecNotSupported	0x80040102	The object that this ROP was called on is not a Folder object.

3.2.5.6 RopMoveCopyMessages

RopMoveCopyMessages moves or copies the specified messages from the source folder to the destination folder.

If the client requests asynchronous execution, the server executes this ROP asynchronously. For more details about RopProgress, including usages, restrictions, and notes, see [MS-OXCPRPT] section 2.2.22.

If any of the messages fail to move or copy as requested, the server reports partial completion.

The following specific error codes apply to this ROP. For more details about ROP errors, see [MS-OXCDATA] section 2.4.

Name	Value	Meaning
ecNotSupported	0x80040102	Either the source or the destination object is not a Folder object.

Name	Value	Meaning
		OR Either the source or the destination object is a search folder.

3.2.5.7 RopMoveFolder

RopMoveFolder moves a folder from one parent to another. All the content and subfolders of the folder are moved with the folder.

If the client requests asynchronous execution, the server executes this ROP asynchronously. For more details about RopProgress, including usages, restrictions, and notes, see [MS-OXCPRPT] section 2.2.22.

The following specific error codes apply to this ROP. For more details about ROP errors, see [MS-OXCDATA] section 2.4.

Name	Value	Meaning
ecNotFound	0x8004010F	There is no folder with the specified ID.
ecNotSupported	0x80040102	Either the source or the destination object is not a Folder object. OR Either the source or the destination object is a search folder.

3.2.5.8 RopCopyFolder

RopCopyFolder creates a new folder on the destination parent folder, copying the properties and content of the source folder to the new folder.

All messages in the source folder MUST be duplicated in the new folder.

If the *WantRecursive* flag is used, the subfolders contained in the source folder are also duplicated in the new folder, including their properties, messages, and subfolders (in a recursive manner).

If the client requests asynchronous execution, the server executes this ROP asynchronously. For more details about RopProgress, including usages, restrictions, and notes, see [MS-OXCPRPT] section 2.2.22.

The following specific error codes apply to this ROP. For more details about ROP errors, see [MS-OXCDATA] section 2.4.

Name	Value	Meaning
ecNotFound	0x8004010F	There is no folder with the specified ID.
ecNotSupported	0x80040102	Either the source or the destination object is not a Folder object. OR Either the source or the destination object is a search folder.

3.2.5.9 RopEmptyFolder

RopEmptyFolder removes all normal (non-FAI) messages and all subfolders from the specified folder.

If the *WantDeleteAssociated* flag is specified, then the server removes all FAI messages, in addition to the **normal messages**, and all subfolders.

The server soft deletes all messages and subfolders that are removed by this ROP.

If the client requests asynchronous execution, the server executes this ROP asynchronously. For more details about RopProgress, including usages, restrictions, and notes, see [MS-OXCPRPT] section 2.2.22.

If the server is unable to remove at least one message or subfolder, the ROP returns nonzero for **PartialCompletion**.

The following specific error codes apply to this ROP. For more details about ROP errors, see [MS-OXCDATA] section 2.4.

Name	Value	Meaning
ecNotSupported	0x80040102	The object that this ROP was called on is not a Folder object.

3.2.5.10 RopHardDeleteMessagesAndSubfolders

RopHardDeleteMessagesAndSubfolders behaves in the same way as RopEmptyFolder, except that messages and subfolders that are removed by this ROP are hard deleted instead of soft deleted.

The following specific error codes apply to this ROP. For more details about ROP errors, see [MS-OXCDATA] section 2.4.

Name	Value	Meaning
ecNotSupported	0x80040102	The object that this ROP was called on is not a Folder object.

3.2.5.11 RopDeleteMessages

RopDeleteMessages removes existing messages from the database.

Messages that are deleted by using this ROP are soft deleted.

If the client requests asynchronous execution, the server executes this ROP asynchronously. For more details about RopProgress, including usages, restrictions, and notes on the asynchronous execution of ROPs, see [MS-OXCPRPT] section 2.2.22.

The following specific error codes apply to this ROP. For more details about ROP errors, see [MS-OXCDATA] section 2.4.

Name	Value	Meaning
ecNotSupported	0x80040102	The object that this ROP was called on is not a Folder object.

3.2.5.12 RopHardDeleteMessages

RopHardDeleteMessages behaves in the same way as RopDeleteMessages, except that the deleted messages are hard deleted instead of soft deleted.

If the client requests asynchronous execution, the server executes this ROP asynchronously. For more details about RopProgress, including usages, restrictions, and notes, see [MS-OXCPRPT] section 2.2.22.

The following specific error codes apply to this ROP. For more details about ROP errors, see [MS-OXCDATA] section 2.4.

Name	Value	Meaning
ecNotSupported	0x80040102	The object that this ROP was called on is not a Folder object.

3.2.5.13 RopGetHierarchyTable

RopGetHierarchyTable returns a Table object on which table operations can be performed. For more details about Table objects and table operations, see [MS-OXCTABL]. The Table object that is returned MUST allow access to the information that is contained in subfolders of the Folder object on which this ROP is executed.

If the Depth flag is specified, this ROP returns a table with all subfolders under the Folder object on which this ROP is executed, including the subfolders of its subfolders (recursively).

If the SuppressesNotifications flag is specified, actions from this client do not trigger events on this table.

The **RowCount** is always returned, but if the DeferredErrors flag is specified, the count might not be correct. The client cannot rely on the values of these response fields if DeferredErrors is set.

If the SoftDeletes flag is specified, the table that is returned provides access to the information that is contained in subfolders that have been soft deleted. If the SoftDeletes flag is not specified, the table that is returned provides access to the information that is contained in subfolders that have not been soft deleted.

The following specific error codes apply to this ROP. For more details about ROP errors, see [MS-OXCDATA] section 2.4.

Name	Value	Meaning
ecNotSupported	0x80040102	The object that this ROP was called on is not a Folder object.

3.2.5.14 RopGetContentsTable

RopGetContentsTable returns a Table object on which table operations can be performed. For more details about Table objects and table operations, see [MS-OXCTABL]. The Table object that is returned provides information about messages that are directly under the Folder object on which this ROP is executed.

If the Associated flag is specified, the table that is returned only contains information about FAI messages that are directly under the specified folder. If the Associated flag is not specified, the table that is returned contains information about only normal (non-FAI) messages that are directly under the specified folder.

If the SoftDeletes flag is specified, the table that is returned provides access to the information about messages that have been soft deleted. If the SoftDeletes flag is not specified, the table that is returned provides access to the information about messages that have not been soft deleted.

If the NoNotifications flag is specified, actions that would normally trigger notifications on the table do not trigger any notifications.

The **RowCount** is always returned, but if the DeferredErrors flag is specified, the count might not be correct. The client cannot rely on the values of these response fields if DeferredErrors is set.

The following specific error codes apply to this ROP. For more details about ROP errors, see [MS-OXCDATA] section 2.4.

Name	Value	Meaning
ecNotSupported	0x80040102	The object that this ROP was called on is not a Folder object.

3.2.6 Timer Events

None.

3.2.7 Other Local Events

None.

4 Protocol Examples

The following examples illustrate the byte order of ROPs in a buffer being prepared for transmission. Note that the examples in this section show only the relevant portions of the specified ROPs; this is not the final byte sequence that gets transmitted over the wire. Also note that the data format for a multi-byte field appears in **little-endian** format, with the **bytes** in the field presented from least significant to most significant.

Frequently, these ROP requests are packed with other ROP requests, compressed, obfuscated, and then packed in one or more **remote procedure call (RPC)** calls. These examples assume that the client has already successfully logged on to the server and has obtained any **Server object handles** that are to be used as inputs into the ROPs. For more information about RPC calls, see [MS-OXCRPC].

Examples in this section use the following format for byte sequences:

```
0080: 45 4d 53 4d 44 42 2e 44-4c 4c 00 00 00 00 00 00
```

The bold value at the far left ("**0080**") is the offset of the following bytes into the buffer, expressed in hexadecimal notation. Following the offset is a series of up to 16 bytes, with each two-character sequence describing the value of one byte in hexadecimal notation. The bolded byte "**4d**" (01001101) is located 0x83 bytes (131 bytes) from the beginning of the buffer. The dash between the eighth byte ("44") and the ninth byte ("4c") bytes has no semantic value, and serves only to distinguish the eight-byte boundary for readability.

This byte sequence is followed by one or more lines that interpret it. In larger examples, the byte sequence is shown once in its entirety and then repeated in smaller chunks, with each smaller chunk interpreted separately.

When explaining *InputHandleIndex* values, the example text describes the **Server object** that is referenced by the handle index. For more information about Server object handles, see [MS-OXCROPS] section 1.3.1.

4.1 Creating a New Folder by Using RopCreateFolder

The following example describes the content of the ROP request buffer and ROP response buffer for a successful RopCreateFolder operation, as specified in section 2.2.2.

4.1.1 Client Request Buffer

The client request buffer for the RopCreateFolder example is formatted as follows:

```
0000: 1c 00 00 01 01 01 00 00-46 00 6f 00 6c 00 64 00
0010: 65 00 72 00 31 00 00 00-00 00
```

The first four bytes refer to the **RopId**, **LogonId**, **InputHandleIndex**, and **OutputHandleIndex** fields of the RopCreateFolder format, as specified in section 2.2.2.

```
0000: 1c 00 00 01
```

RopId: "0x1c" (RopCreateFolder)

LogonId: "0x00"

InputHandleIndex: "0x00" This value specifies the location where the handle for the input folder is stored.

OutputHandleIndex: "0x01". This value specifies the location where the handle for the newly created folder is stored.

The next four bytes contain the **FolderType**, **UseUnicodeStrings**, **OpenExisting**, and **Reserved** fields of the RopCreateFolder format, as specified in section 2.2.2. These fields affect how the operation is carried out.

```
0004: 01 01 00 00
```

FolderType: "0x01" (generic). The folder is a generic folder.

UseUnicodeStrings: "0x01" (TRUE). This value indicates that the folder name is in Unicode format.

OpenExisting: "0x00" (FALSE). This value indicates that the operation will fail if the folder already exists.

Reserved: "0x00" (FALSE). This value indicates that the **LongTermID** field is not included in the request.

The next 16 bytes contain the **DisplayName** field. This field is a null-terminated string that contains the name of the folder to create, and is formatted as Unicode text, as indicated by the value sent in the **UseUnicodeStrings** field.

```
0008: 46 00 6f 00 6c 00 64 00-65 00 72 00 31 00 00 00
```

DisplayName: "Folder1"

The **Comment** field is sent next and, in this example, is a null-terminated string that consists of zero (0) characters, and follows the same text format (Unicode) as the **DisplayName** field.

```
0018: 00 00
```

Comment: ""

4.1.2 Server Responds to Client Request

```
0000: 1c 01 00 00 00 00 01 00-00 00 0e 91 52 12 00
```

The first six bytes contain the **RopId**, **OutputHandleIndex**, and **ReturnValue** response fields, as specified in section 2.2.2.2:

```
0000: 1c 01 00 00 00 00
```

RopId: "0x1c" (RopCreateFolder)

OutputHandleIndex: "0x01". This is the same index as the **OutputHandleIndex** specified in the request.

ReturnValue: "0x00000000". This response indicates that the folder has successfully been created.

The next eight bytes provide the **FolderId** property for the newly created folder:

```
0006: 01 00 00 00 0e 91 52 12
```

FolderId: 0001-00000e915212

The next byte contains the **IsExistingFolder** response field.

```
000F: 00
```

IsExistingFolder: "0x00" (FALSE). This value indicates that a new folder was created.

Because **IsExistingFolder** is FALSE, this is the last byte of this ROP response buffer.

4.2 Deleting an Existing Folder by Using RopDeleteFolder

The following example describes the content of the ROP request buffer and ROP response buffer for a successful RopDeleteFolder operation, as specified in section 2.2.3.

4.2.1 Client Request Buffer

The client request buffer for the RopDeleteFolder example consists of a 12-byte sequence, formatted as follows:

```
0000: 1d 00 01 05 01 00 00 00-0e 8e df 36
```

RopId: "0x1d" (RopDeleteFolder)

LogonID: "0x00"

InputHandleIndex: "0x01". This value specifies the location where the handle for the folder is stored.

DeleteFolderFlags: "0x05" (**DEL_MESSAGES** | **DEL_FOLDERS**). This value indicates that the specified folder and all messages and subfolders within the folder have to be deleted.

FolderId: "0001-00000e8edf36". This field uniquely identifies the folder to be deleted.

4.2.2 Server Responds to Client Request

The server response buffer for the successful RopDeleteFolder operation consists of a 7-byte sequence, formatted as follows:

```
0000: 1d 01 00 00 00 00 00
```

RopId: "0x1d" (RopDeleteFolder)

InputHandleIndex: "0x01". This is the same index as the **InputHandleIndex** specified in the request.

ReturnValue: "0x00000000". This response indicates that the folder has successfully been deleted.

PartialCompletion: "0x00" (FALSE). This response indicates that the operation was fully completed and all messages and folders specified in the ROP request were deleted.

4.3 Deleting Messages Within a Folder

The following example describes the content of the ROP request buffer and ROP response buffer for a successful RopDeleteMessages operation, as specified in section 2.2.11. In this example, a folder contains two messages, message ID values for which are passed in the ROP.

4.3.1 Client Request Buffer

The client request buffer for the RopDeleteMessages example consists of the sequence of bytes formatted as follows:

```
0000: 1e 00 00 00 01 02 00 01-00 00 00 0e 8e f1 48 01
0010: 00 00 00 0e 8e c3 02
```

The first five bytes refer to the **RopId**, **LogonID**, **InputHandleIndex**, **WantAsynchronous**, and **NotifyNonRead** fields of the RopDeleteMessages format, as specified in section 2.2.11.

```
0000: 1e 00 00 00 01
```

RopId: "0x1e" (RopDeleteMessages)

LogonID: "0x00"

InputHandleIndex: "0x00". This value specifies the location where the handle for the messages' parent folder is stored.

WantAsynchronous: "0x00" (FALSE).The ROP is executed synchronously.

NotifyNonRead: "0x01" (TRUE). The client wants a notification if a message was deleted before it was read.

The remaining bytes in the buffer consist of the list of messages to delete.

```
0005: 02 00 01 00 00 00 0e 8e-f1 48 01 00 00 00 0e 8e
0015: c3 02
```

MessageIdCount: "0x0002". This value indicates how many messages are listed for deletion in the **MessageIds** field.

MessageIds:

"0001-00000e8ef148". MID of a message to be deleted.

"0001-00000e8ec302". MID of a message to be deleted.

4.3.2 Server Responds to Client Request

The server response buffer for the successful RopDeleteMessages operation consists of a 7-byte sequence, formatted as follows:

```
0000: 1e 00 00 00 00 00 00
```

RopId: "0x1e" (RopDeleteMessages)

InputHandleIndex: "0x00". This is the same index as the **InputHandleIndex** that is specified in the request buffer.

ReturnValue: "0x00000000". This response indicates that the items were successfully deleted.

PartialCompletion: "0x00" (FALSE). This response indicates that the operation was fully completed and all messages that were specified in the ROP request were deleted.

4.4 Moving Messages From One Folder to Another

The following example describes the content of the ROP request buffer and ROP response buffer for a successful RopMoveCopyMessages operation, as specified in section 2.2.6. In this example, a message, specified by its MID, is moved from one folder to another, specified by folder handles.

4.4.1 Client Request Buffer

The client request buffer for the RopMoveCopyMessages example consists of the sequence of bytes formatted as follows:

```
0000: 33 00 00 01 01 00 01 00-00 00 0e 8e ec 5d 00 00
```

The first four bytes refer to the **RopId**, **LogonID**, **SourceHandleIndex**, and **DestHandleIndex** fields of the RopMoveCopyMessages format, as specified in section 2.2.6.

```
0000: 33 00 00 01
```

RopId: "0x33" (RopMoveCopyMessages)

LogonID: "0x00"

SourceHandleIndex: "0x00". This value specifies the location where the handle for the messages' parent folder is stored.

DestHandleIndex: "0x01". This value specifies the location where the handle for the destination folder is stored.

The following 10 bytes consist of the list of messages to move.

```
0004: 01 00 01 00 00 00 0e 8e-ec 5d
```

MessageIdCount: "0x0001". This value indicates how many messages are listed for moving in the **MessageIds** field.

MessageIds: "0001-00000e8eec5d". Message ID of the message to be moved.

The final two bytes in the buffer contain the **WantAsynchronous** and **WantCopy** fields.

```
000e: 00 00
```

WantAsynchronous: 0x00 (FALSE). The ROP is executed synchronously.

WantCopy: 0x00 (FALSE). This value indicates that the operation is a move rather than a copy.

4.4.2 Server Responds to Client Request

The server response buffer for the successful RopMoveCopyMessages operation consists of a 7-byte sequence formatted as follows:

```
0000: 33 00 00 00 00 00 00
```

RopId: "0x33" (RopMoveCopyMessages)

SourceHandleIndex: "0x00". This is the same index as the **SourceHandleIndex** that is specified in the request buffer.

ReturnValue: "0x00000000". This response indicates that the items were successfully moved.

PartialCompletion: "0x00" (FALSE). This response indicates that the operation was fully completed and all messages specified in the ROP request were moved.

4.5 Moving a Folder

The following example describes the content of the ROP request buffer and ROP response buffer for a successful RopMoveFolder operation, as specified in section 2.2.7. In this example, a folder, specified by its FID, is moved to a new location in the folder hierarchy.

4.5.1 Client Request Buffer

The client request buffer for the RopMoveFolder example consists of a 30-byte sequence formatted as follows:

```
0000: 35 00 01 02 01 01 01 00-00 00 0e 8e df 36 46 00
0010: 6f 00 6c 00 64 00 65 00-72 00 31 00 00 00
```

The first six bytes of the request buffer map to the **RopId**, **LogonID**, **SourceHandleIndex**, **DestHandleIndex**, **WantAsynchronous**, and **UseUnicode** fields of the RopMoveFolder format, as specified in section 2.2.7.

```
0000: 35 00 01 02 01 01
```

RopId: "0x35" (RopMoveFolder)

LogonID: "0x00"

SourceHandleIndex: "0x01". This value specifies the location where the handle for the parent folder of the folder to move is stored.

DestHandleIndex: "0x02". This value specifies the location where the handle for the destination folder is located.

WantAsynchronous: "0x01" (TRUE). The ROP is executed asynchronously.

UseUnicode: "0x01" (TRUE). This value indicates that the **NewFolderName** field is in Unicode format.

The next eight bytes are the **FolderId** field.

```
0006: 01 00 00 00 0e 8e df 36
```

FolderId: "0001-00000e8edf36"

The remaining 16 bytes of the request buffer specify the new name of the folder.

```
000e: 46 00 6f 00 6c 00 64 00-65 00 72 00 31 00 00 00
```

NewFolderName: "Folder1"

4.5.2 Server Responds to Client Request

The server response buffer for the successful RopMoveFolder operation consists of a 7-byte sequence, formatted as follows:

```
0000: 35 01 00 00 00 00 00
```

RopId: "0x35" (RopMoveFolder)

SourceHandleIndex: "0x01". This is the same index as the **SourceHandleIndex** specified in the request buffer.

ReturnValue: "0x00000000". This response indicates that the folder was successfully moved.

PartialCompletion: "0x00" (FALSE). This response indicates that the operation was fully completed.

4.6 Copying a Folder

The following example describes the content of the ROP request buffer and ROP response buffer for a successful RopCopyFolder operation, as specified in section 2.2.8. In this example, a folder, specified by its FID, is then copied to a new location in the folder hierarchy.

4.6.1 Client Request Buffer

The client request buffer for the RopCopyFolder example consists of a sequence of bytes, formatted as follows:

```
0000: 36 00 00 01 01 01 01 01-00 00 00 0e 8e df 36 46
0010: 00 6f 00 6c 00 64 00 65-00 72 00 31 00 00 00
```

The first seven bytes of the request buffer map to the **RopId**, **LogonID**, **SourceHandleIndex**, **DestHandleIndex**, **WantAsynchronous**, **WantRecursive**, and **UseUnicode** fields of the RopCopyFolder format, as specified in section 2.2.8.

```
0000: 36 00 00 01 01 01 01
```

RopId: "0x36" (RopCopyFolder)

LogonID: "0x00"

SourceHandleIndex: "0x00". This value specifies the location of where the handle for the parent folder of the folder to copy is stored.

DestHandleIndex: "0x01". This value specifies the location where the handle for the destination folder is stored.

WantAsynchronous: "0x01" (TRUE). The ROP is executed asynchronously.

WantRecursive: "0x01" (TRUE). The operation recursively copies all subfolders, messages, and properties.

UseUnicode: "0x01" (TRUE). This value indicates that the **NewFolderName** argument is in Unicode format.

The next eight bytes are the **FolderId** field.

```
0006: 01 00 00 00 0e 8e df 36
```

FolderId: "0001-00000e8edf36"

The remaining 16 bytes of the request buffer specify the new name of the folder.

```
000e: 46 00 6f 00 6c 00 64 00-65 00 72 00 31 00 00 00
```

NewFolderName: "Folder1"

4.6.2 Server Responds to Client Request

The server response buffer for the successful RopCopyFolder operation consists of a 7-byte sequence, formatted as follows:

```
0000: 36 00 00 00 00 00 00
```

RopId: "0x36" (RopCopyFolder)

SourceHandleIndex: "0x00". This is the same index as the **SourceHandleIndex** specified in the request buffer.

ReturnValue: "0x00000000". This response indicates that the folder was successfully moved.

PartialCompletion: "0x00" (FALSE). This response indicates that the operation was fully completed.

4.7 Getting the List of Subfolders Within a Message Folder

This example shows what the buffer for a successful RopGetHierarchyTable call looks like, as specified in section 2.2.13. For more details about tables, see [MS-OXCTABL].

4.7.1 Client Request Buffer

The client request buffer for the RopGetHierarchyTable example consists of a 5-byte sequence, formatted as follows:

```
0000: 04 00 01 02 00
```

RopId: "0x04" (RopGetHierarchyTable)

LogonID: "0x00"

InputHandleIndex: "0x01". This value specifies the location where the handle for the folder to retrieve the hierarchy table is stored.

OutputHandleIndex: "0x02". This value specifies the location where the handle for the hierarchy table will be stored.

TableFlags: "0x00". For details about **InputHandleIndex** values, see section 2.2.13.1.3.

4.7.2 Server Responds to Client Request

The server response buffer for the successful RopGetHierarchyTable operation consists of a 10-byte sequence, formatted as follows:

```
0000: 04 02 00 00 00 00 15 00-00 00
```

RopId: "0x04" (RopGetHierarchyTable)

OutputHandleIndex: "0x02". This is the same index as the **OutputHandleIndex** that is specified in the request buffer.

Return Value: "0x00000000". This response indicates that the hierarchy table was retrieved.

RowCount: "0x00000015". The table contains 21 rows.

4.8 Setting the Search Criteria for a Search Folder

This example illustrates the buffer contents for a successful RopSetSearchCriteria operation, as specified in section 2.2.4. The search folder is referred to by the **InputHandleIndex** parameter, and the search criteria filter specifies restrictions that limit the items in the search folder — in this case, mail items for which the PidTagImportance property is set to "0x00000002" (High). For more details about the structure of a restriction, see [MS-OXCDATA] section 2.13.

4.8.1 Client Request Buffer

The client request buffer for the RopSetSearchCriteria example operation consists of a 316-byte sequence, formatted as follows:

```
0000: 30 00 01 29 01 00 02 00-00 07 00 02 03 02 00 01
0010: 00 1F 00 1A 00 1F 00 1A-00 49 00 50 00 4D 00 2E
0020: 00 41 00 70 00 70 00 6F-00 69 00 6E 00 74 00 6D
0030: 00 65 00 6E 00 74 00 00-00 02 03 02 00 01 00 1F
0040: 00 1A 00 1F 00 1A 00 49-00 50 00 4D 00 2E 00 43
0050: 00 6F 00 6E 00 74 00 61-00 63 00 74 00 00 00 02
0060: 03 02 00 01 00 1F 00 1A-00 1F 00 1A 00 49 00 50
0070: 00 4D 00 2E 00 44 00 69-00 73 00 74 00 4C 00 69
0080: 00 73 00 74 00 00 00 02-03 02 00 01 00 1F 00 1A
0090: 00 1F 00 1A 00 49 00 50-00 4D 00 2E 00 41 00 63
00A0: 00 74 00 69 00 76 00 69-00 74 00 79 00 00 00 02
00B0: 03 02 00 01 00 1F 00 1A-00 1F 00 1A 00 49 00 50
00C0: 00 4D 00 2E 00 53 00 74-00 69 00 63 00 6B 00 79
00D0: 00 4E 00 6F 00 74 00 65-00 00 00 02 03 00 00 01
00E0: 00 1F 00 1A 00 1F 00 1A-00 49 00 50 00 4D 00 2E
00F0: 00 54 00 61 00 73 00 6B-00 00 00 02 03 02 00 01
0100: 00 1F 00 1A 00 1F 00 1A-00 49 00 50 00 4D 00 2E
0110: 00 54 00 61 00 73 00 6B-00 2E 00 00 00 00 01 00
0120: 04 04 03 00 17 00 03 00-17 00 02 00 00 00 01 00
0130: 01 00 00 00 00 00 14 88-2A 00 02 00
```

The first three bytes of the request buffer map to the **RopId**, **LogonID**, and **InputHandleIndex** fields of the RopSetSearchCriteria format, as specified in section 2.2.4.

```
0000: 30 00 01
```

RopId: "0x30" (RopSetSearchCriteria)

LogonID: "0x00"

InputHandleIndex: "0x01". This value specifies the location where the handle for the search folder to configure is stored.

The next 299 bytes comprise the restriction that defines the search criteria for the search folder, broken down in further detail as follows:

```
0003: 29 01 00 02 00 00 07 00-02 03 02 00 01 00 1F 00
0013: 1A 00 1F 00 1A 00 49 00-50 00 4D 00 2E 00 41 00
0023: 70 00 70 00 6F 00 69 00-6E 00 74 00 6D 00 65 00
0033: 6E 00 74 00 00 00 02 03-02 00 01 00 1F 00 1A 00
```

```
0043: 1F 00 1A 00 49 00 50 00-4D 00 2E 00 43 00 6F 00
0053: 6E 00 74 00 61 00 63 00-74 00 00 00 02 03 02 00
0063: 01 00 1F 00 1A 00 1F 00-1A 00 49 00 50 00 4D 00
0073: 2E 00 44 00 69 00 73 00-74 00 4C 00 69 00 73 00
0083: 74 00 00 00 02 03 02 00-01 00 1F 00 1A 00 1F 00
0093: 1A 00 49 00 50 00 4D 00-2E 00 41 00 63 00 74 00
00A3: 69 00 76 00 69 00 74 00-79 00 00 00 02 03 02 00
00B3: 01 00 1F 00 1A 00 1F 00-1A 00 49 00 50 00 4D 00
00C3: 2E 00 53 00 74 00 69 00-63 00 6B 00 79 00 4E 00
00D3: 6F 00 74 00 65 00 00 00-02 03 00 00 01 00 1F 00
00E3: 1A 00 1F 00 1A 00 49 00-50 00 4D 00 2E 00 54 00
00F3: 61 00 73 00 6B 00 00 00-02 03 02 00 01 00 1F 00
0103: 1A 00 1F 00 1A 00 49 00-50 00 4D 00 2E 00 54 00
0113: 61 00 73 00 6B 00 2E 00-00 00 00 01 00 04 04 03
0123: 00 17 00 03 00 17 00 02-00 00 00
```

RestrictionDataSize: "0x0129". This value specifies that the size of the restriction block is 297 bytes.

RestrictionData: Bytes "0005-012A", which translate into the following restriction:

RestrictType: "0x00" (RES_AND)

RestrictCount: "0x0002"

RestrictType: "0x00" (RES_AND)

RestrictCount: "0x0007"

RestrictType: "0x02" (RES_NOT)

RestrictType: "0x03" (RES_CONTENT)

FuzzyLevel: "0x00010002" (FL_PREFIX | FL_IGNORECASE)

PropTag1: "0x001a001f" (PidTagMessageClass)

PropTag2: "0x001a001f" (PidTagMessageClass)

PropRule: "IPM.Appointment"

RestrictType: "0x02" (RES_NOT)

RestrictType: "0x03" (RES_CONTENT)

FuzzyLevel: "0x00010002" (FL_PREFIX | FL_IGNORECASE)

PropTag1: "0x001a001f" (PidTagMessageClass)

PropTag2: "0x001a001f" (PidTagMessageClass)

PropRule: "IPM.Contact"

RestrictType: "0x02" (RES_NOT)

RestrictType: "0x03" (RES_CONTENT)

FuzzyLevel: "0x00010002" (FL_PREFIX | FL_IGNORECASE)

PropTag1: "0x001a001f" (PidTagMessageClass)

PropTag2: "0x001a001f" (PidTagMessageClass)

PropRule: "IPM.DistList"

RestrictType: "0x02" (RES_NOT)

RestrictType: "0x03" (RES_CONTENT)

FuzzyLevel: "0x00010002" (FL_PREFIX | FL_IGNORECASE)

PropTag1: "0x001a001f" (PidTagMessageClass)

PropTag2: "0x001a001f" (PidTagMessageClass)

PropRule: "IPM.Activity"

RestrictType: "0x02" (RES_NOT)

RestrictType: "0x03" (RES_CONTENT)

FuzzyLevel: "0x00010002" (FL_PREFIX | FL_IGNORECASE)

PropTag1: "0x001a001f" (PidTagMessageClass)

PropTag2: "0x001a001f" (PidTagMessageClass)

PropRule: "IPM.StickyNote"

RestrictType: "0x02" (RES_NOT)

RestrictType: "0x03" (RES_CONTENT)

FuzzyLevel: "0x00010000" (FL_FULLSTRING | FL_IGNORECASE)

PropTag1: "0x001a001f" (PidTagMessageClass)

PropTag2: "0x001a001f" (PidTagMessageClass)

PropRule: "IPM.Task"

RestrictType: "0x02" (RES_NOT)

RestrictType: "0x03" (RES_CONTENT)

FuzzyLevel: "0x00010002" (FL_PREFIX | FL_IGNORECASE)

PropTag1:"0x001a001f" (PidTagMessageClass)

PropTag2: "0x001a001f" (PidTagMessageClass)

PropRule: "IPM.Task. "

RestrictType: "0x00" (RES_AND)

RestrictCount: "0x0001"

RestrictType: "0x04" (RES_PROPERTY)

RelOp: "0x04" (RELOP_EQ)

PropTag1: "0x00170003" (PidTagImportance)

PropTag2: "0x00170003" (PidTagImportance)

PropValue: "0x00000002"

A shorthand description of the restriction is as follows:

((PidTagMessageClass is not equal to "IPM.Appointment" AND

PidTagMessageClass is not equal to "IPM.Contact" AND

PidTagMessageClass is not equal to "IPM.DistList" AND

PidTagMessageClass is not equal to "IPM.Activity" AND

PidTagMessageClass is not equal to "IPM.StickyNote" AND

PidTagMessageClass is not equal to "IPM.Task" AND

PidTagMessageClass is not equal to "IPM.Task.")

AND

(PidTagItemTemporaryflags bit 0x00000001 is not equal to 0 (zero)))

The next 10 bytes consist of the **FolderIdCount** and **FolderIds** fields:

```
012E: 01 00 01 00 00 00 00 00-14 88
```

FolderIdCount: "0x0001". This value specifies the number of folders within the scope of the search folder.

FolderIds: "0001-000000001488". Identifies the folder to be searched.

The remaining four bytes represent the **SearchFlags** field.

```
0138: 2A 00 02 00
```

SearchFlags: "0x0002002A" (RESTART _ SEARCH | SHALLOW _ SEARCH | BACKGROUND _ SEARCH | NON _ CONTENT _ INDEXED _ SEARCH)

4.8.2 Server Responds to Client Request

The server response buffer for the RopSetSearchCriteria operation consists of a 6-byte sequence, formatted as follows:

```
0000: 30 01 00 00 00 00
```

RopId: "0x30" (RopSetSearchCriteria)

InputHandleIndex: "0x01". This is the same index as the **InputHandleIndex** that was specified in the request buffer.

ReturnValue: "0x00000000". This response indicates that the search criteria were set on the folder.

4.9 Getting the Search Criteria for a Search Folder

This example illustrates the buffer contents for a successful RopGetSearchCriteria operation, as specified in section 2.2.5. The search folder is referred to by the *InputHandleIndex* parameter, and the search criteria filter that is returned specifies restrictions that limit the items in the search folder – in this case, mail items for which the PidTagImportance property is set to 0x00000002 (High). For more details about the structure of a restriction, see [MS-OXCDATA] section 2.13.

4.9.1 Client Request Buffer

The client request buffer for the RopGetSearchCriteria example consists of a sequence of bytes, formatted as follows:

```
0000: 31 00 00 01 01 00
```

RopId: "0x31" (RopGetSearchCriteria)

LogonID: "0x00"

InputHandleIndex: "0x00". This value specifies the location where the handle for the search folder to query for criteria is stored.

UseUnicode: "0x01" (TRUE). This value indicates that the response restriction is expected to include Unicode strings.

IncludeRestriction: "0x01". This value indicates that the server response is expected to include the restriction data for the search folder.

IncludeFolders: "0x00". This value indicates that the server response is not expected to include the set of folders within the search scope.

4.9.2 Server Responds to Client Request

The server response buffer for the successful RopGetSearchCriteria operation consists of a 312-byte sequence, formatted as follows:

```
0000: 31 00 00 00 00 00 29 01-00 02 00 00 07 00 02 03
0010: 02 00 01 00 1F 00 1A 00-1F 00 1A 00 49 00 50 00
0020: 4D 00 2E 00 41 00 70 00-70 00 6F 00 69 00 6E 00
0030: 74 00 6D 00 65 00 6E 00-74 00 00 00 02 03 02 00
0040: 01 00 1F 00 1A 00 1F 00-1A 00 49 00 50 00 4D 00
0050: 2E 00 43 00 6F 00 6E 00-74 00 61 00 63 00 74 00
0060: 00 00 02 03 02 00 01 00-1F 00 1A 00 1F 00 1A 00
0070: 49 00 50 00 4D 00 2E 00-44 00 69 00 73 00 74 00
0080: 4C 00 69 00 73 00 74 00-00 00 02 03 02 00 01 00
0090: 1F 00 1A 00 1F 00 1A 00-49 00 50 00 4D 00 2E 00
00A0: 41 00 63 00 74 00 69 00-76 00 69 00 74 00 79 00
00B0: 00 00 02 03 02 00 01 00-1F 00 1A 00 1F 00 1A 00
00C0: 49 00 50 00 4D 00 2E 00-53 00 74 00 69 00 63 00
```

```
00D0: 6B 00 79 00 4E 00 6F 00-74 00 65 00 00 00 02 03
00E0: 00 00 01 00 1F 00 1A 00-1F 00 1A 00 49 00 50 00
00F0: 4D 00 2E 00 54 00 61 00-73 00 6B 00 00 00 02 03
0100: 02 00 01 00 1F 00 1A 00-1F 00 1A 00 49 00 50 00
0110: 4D 00 2E 00 54 00 61 00-73 00 6B 00 2E 00 00 00
0120: 00 01 00 04 04 03 00 17-00 03 00 17 00 02 00 00
0130: 00 00 00 00 01 00 00 00
```

The first six bytes contain the **RopId**, **InputHandleIndex**, and **ReturnValue** response fields specified in section 2.2.6.2:

```
0000: 31 00 00 00 00 00
```

RopId: "0x31" (RopGetSearchCriteria)

InputHandleIndex: "0x00". This is the same index as the **InputHandleIndex** specified in the request buffer.

ReturnValue: "0x00000000". This response indicates that the search criteria for the search folder were successfully retrieved.

The next 299 bytes comprise the restrictions that define the search criteria for the search folder, broken down in further detail as follows:

```
0006: 29 01 00 02 00 00 07 00-02 03 02 00 01 00 1F 00
0016: 1A 00 1F 00 1A 00 49 00-50 00 4D 00 2E 00 41 00
0026: 70 00 70 00 6F 00 69 00-6E 00 74 00 6D 00 65 00
0036: 6E 00 74 00 00 00 02 03-02 00 01 00 1F 00 1A 00
0046: 1F 00 1A 00 49 00 50 00-4D 00 2E 00 43 00 6F 00
0056: 6E 00 74 00 61 00 63 00-74 00 00 00 02 03 02 00
0066: 01 00 1F 00 1A 00 1F 00-1A 00 49 00 50 00 4D 00
0076: 2E 00 44 00 69 00 73 00-74 00 4C 00 69 00 73 00
0086: 74 00 00 00 02 03 02 00-01 00 1F 00 1A 00 1F 00
0096: 1A 00 49 00 50 00 4D 00-2E 00 41 00 63 00 74 00
00A6: 69 00 76 00 69 00 74 00-79 00 00 00 02 03 02 00
00B6: 01 00 1F 00 1A 00 1F 00-1A 00 49 00 50 00 4D 00
00C6: 2E 00 53 00 74 00 69 00-63 00 6B 00 79 00 4E 00
00D6: 6F 00 74 00 65 00 00 00-02 03 00 00 01 00 1F 00
00E6: 1A 00 1F 00 1A 00 49 00-50 00 4D 00 2E 00 54 00
00F6: 61 00 73 00 6B 00 00 00-02 03 02 00 01 00 1F 00
0106: 1A 00 1F 00 1A 00 49 00-50 00 4D 00 2E 00 54 00
0116: 61 00 73 00 6B 00 2E 00-00 00 00 01 00 04 04 03
0126: 00 17 00 03 00 17 00 02-00 00 00
```

RestrictionDataSize: "0x0129". This value specifies the size of the restriction block that is 297 bytes.

RestrictionData: Bytes "0008-0130", which translate into the following restriction:

RestrictType: "0x00" (RES_AND)

RestrictCount: "0x0002"

RestrictType: "0x00" (RES_AND)

RestrictCount: "0x0007"

RestrictType: "0x02" (RES_NOT)

RestrictType: "0x03" (RES_CONTENT)

FuzzyLevel: "0x00010002" (FL_PREFIX | FL_IGNORECASE)

PropTag1: "0x001a001f" (PidTagMessageClass)

PropTag2: "0x001a001f" (PidTagMessageClass)

PropRule: "IPM.Appointment"

RestrictType: "0x02" (RES_NOT)

RestrictType: "0x03" (RES_CONTENT)

FuzzyLevel: "0x00010002" (FL_PREFIX | FL_IGNORECASE)

PropTag1: "0x001a001f" (PidTagMessageClass)

PropTag2: "0x001a001f" (PidTagMessageClass)

PropRule: "IPM.Contact"

RestrictType: "0x02" (RES_NOT)

RestrictType: "0x03" (RES_CONTENT)

FuzzyLevel: "0x00010002" (FL_PREFIX | FL_IGNORECASE)

PropTag1: "0x001a001f" (PidTagMessageClass)

PropTag2: "0x001a001f" (PidTagMessageClass)

PropRule: "IPM.DistList"

RestrictType: "0x02" (RES_NOT)

RestrictType: "0x03" (RES_CONTENT)

FuzzyLevel: "0x00010002" (FL_PREFIX | FL_IGNORECASE)

PropTag1: "0x001a001f" (PidTagMessageClass)

PropTag2: "0x001a001f" (PidTagMessageClass)

PropRule: "IPM.Activity"

RestrictType: "0x02" (RES_NOT)

RestrictType: "0x03" (RES_CONTENT)

FuzzyLevel: "0x00010002" (FL_PREFIX | FL_IGNORECASE)

PropTag1: "0x001a001f" (PidTagMessageClass)

PropTag2: "0x001a001f" (PidTagMessageClass)

PropRule: "IPM.StickyNote"

RestrictType: "0x02" (RES_NOT)

RestrictType: "0x03" (RES_CONTENT)

FuzzyLevel: "0x00010000" (FL_FULLSTRING | FL_IGNORECASE)

PropTag1: "0x001a001f" (PidTagMessageClass)

PropTag2: "0x001a001f" (PidTagMessageClass)

PropRule: "IPM.Task"

RestrictType: "0x02" (RES_NOT)

RestrictType: "0x03" (RES_CONTENT)

FuzzyLevel: "0x00010002" (FL_PREFIX | FL_IGNORECASE)

PropTag1: "0x001a001f" (PidTagMessageClass)

PropTag2: "0x001a001f" (PidTagMessageClass)

PropRule: "IPM.Task."

RestrictType: "0x00" (RES_AND)

RestrictCount: "0x0001"

RestrictType: "0x04" (RES_PROPERTY)

RelOp: "0x04" (RELOP_EQ)

PropTag1: "0x00170003" (PidTagImportance)

PropTag2: "0x00170003" (PidTagImportance)

PropValue: "0x00000002"

A shorthand pseudocode description of the restriction is as follows:

((PidTagMessageClass is not equal to "IPM.Appointment" AND

PidTagMessageClass is not equal to "IPM.Contact" AND

PidTagMessageClass is not equal to "IPM.DistList" AND

PidTagMessageClass is not equal to "IPM.Activity" AND

PidTagMessageClass is not equal to "IPM.StickyNote" AND

PidTagMessageClass is not equal to "IPM.Task" AND

PidTagMessageClass is not equal to "IPM.Task. ")

AND

(PidTagItemTemporaryflags bit "0x00000001" is not equal to 0 (zero)))

The final seven bytes of the server response buffer contain the **LogonID**, **FolderIdCount**, and **SearchFlags** fields.

```
0131: 00 00 00 01 00 00 00
```

LogonID: "0x00". This is the same value as the **LogonID** in request buffer.

FolderIdCount: "0x0000".

SearchFlags: "0x00000001" (SEARCH_RUNNING)

5 Security

5.1 Security Considerations for Implementers

There are no special security considerations specific to this specification. General security considerations that pertain to the underlying ROP-based transport apply.

5.2 Index of Security Parameters

None.

6 Appendix A: Product Behavior

The information in this specification is applicable to the following Microsoft products:

- Microsoft® Office Outlook® 2003
- Microsoft® Exchange Server 2003
- Microsoft® Office Outlook® 2007
- Microsoft® Exchange Server 2007
- Microsoft® Outlook® 2010
- Microsoft® Exchange Server 2010

Exceptions, if any, are noted below. If a service pack number appears with the product version, behavior changed in that service pack. The new behavior also applies to subsequent service packs of the product unless otherwise specified. If a product edition appears with the product version, behavior is different in that product edition.

Unless otherwise specified, any statement of optional behavior in this specification prescribed using the terms SHOULD or SHOULD NOT implies product behavior in accordance with the SHOULD or SHOULD NOT prescription. Unless otherwise specified, the term MAY implies that product does not follow the prescription.

<1> Section 2.2: Exchange 2010 does not support the following ROPs when client connection services are deployed on an Exchange server that does not also have a mailbox store installed: RopCopyToStream, RopGetStatus, RopGetStoreState, RopLockRegionStream, RopRegisterSynchronizationNotifications, RopSeekRowFractional, RopSetSynchronizationNotificationGuid, RopSynchronizationOpenAdvisor, RopUnlockRegionStream.

<2> Section 2.2.1.2.2: Exchange 2003 and Exchange 2007 return a value of FALSE for the *HasRules* parameter, even when there are rules on the Inbox.

<3> Section 2.2.2.2.3: The initial version of Exchange 2010 and Exchange 2010 Service Pack 1 return 0x00 (FALSE) for the *IsExistingFolder* parameter when the named folder already exists.

<4> Section 2.2.4.1.6: Exchange 2007 does not return InvalidParameter for unrecognized bit values, but silently ignores them.

<5> Section 2.2.4.2.1: Exchange 2007 returns ecNone instead of ecSearchFolderScopeViolation for the RopSetSearchCriteria operation.

<6> Section 2.2.6: Exchange 2010 does not support RopProgress.

<7> Section 2.2.13.1.3: Exchange 2007 does not fail RopGetHierarchyTable when undefined flags are set, but instead silently ignores them.

<8> Section 2.2.14.1.3: Exchange 2010 fails **RopGetContentsTable** with the error code NotSupported if the Associated bit is set.

<9> Section 2.2.14.1.3: Exchange 2010 fails **RopGetContentsTable** with the error code NotSupported when the UseUnicode bit is set.

<10> Section 2.2.14.1.3: Exchange 2007 silently ignores undefined bits instead of failing the operation.

7 Change Tracking

This section identifies changes that were made to the [MS-OXCFOLD] protocol document between the May 2010 and August 2010 releases. Changes are classified as New, Major, Minor, Editorial, or No change.

The revision class **New** means that a new document is being released.

The revision class **Major** means that the technical content in the document was significantly revised. Major changes affect protocol interoperability or implementation. Examples of major changes are:

- A document revision that incorporates changes to interoperability requirements or functionality.
- An extensive rewrite, addition, or deletion of major portions of content.
- The removal of a document from the documentation set.
- Changes made for template compliance.

The revision class **Minor** means that the meaning of the technical content was clarified. Minor changes do not affect protocol interoperability or implementation. Examples of minor changes are updates to clarify ambiguity at the sentence, paragraph, or table level.

The revision class **Editorial** means that the language and formatting in the technical content was changed. Editorial changes apply to grammatical, formatting, and style issues.

The revision class **No change** means that no new technical or language changes were introduced. The technical content of the document is identical to the last released version, but minor editorial and formatting changes, as well as updates to the header and footer information, and to the revision summary, may have been made.

Major and minor changes can be described further using the following change types:

- New content added.
- Content updated.
- Content removed.
- New product behavior note added.
- Product behavior note updated.
- Product behavior note removed.
- New protocol syntax added.
- Protocol syntax updated.
- Protocol syntax removed.
- New content added due to protocol revision.
- Content updated due to protocol revision.
- Content removed due to protocol revision.
- New protocol syntax added due to protocol revision.

- Protocol syntax updated due to protocol revision.
- Protocol syntax removed due to protocol revision.
- New content added for template compliance.
- Content updated for template compliance.
- Content removed for template compliance.
- Obsolete document removed.

Editorial changes are always classified with the change type "Editorially updated."

Some important terms used in the change type descriptions are defined as follows:

- **Protocol syntax** refers to data elements (such as packets, structures, enumerations, and methods) as well as interfaces.
- **Protocol revision** refers to changes made to a protocol that affect the bits that are sent over the wire.

The changes made to this document are listed in the following table. For more information, please contact protocol@microsoft.com.

Section	Tracking number (if applicable) and description	Major change (Y or N)	Change type
1.1 Glossary	56541 Added "Table object" to the list of terms that are defined in [MS-OXGLOS].	N	Content update.
1.1 Glossary	57302 Added "Server object table" to the list of terms that are defined in [MS-OXGLOS].	N	Content update.
1.2.1 Normative References	55751 Moved [MS-OXGLOS] from Normative References section to Informative References section.	N	Content update.
2.2.2.1.1 InputHandleIndex	57122 Clarified that this parameter represents the parent folder of the folder to be created.	N	Content update.
2.2.2.2.3 IsExistingFolder	Removed statement referencing "the following parameters."	N	Content update.
2.2.2.2.3 IsExistingFolder	55229 Added a product behavior note specifying that the initial version of Exchange 2010 and Exchange 2010 Service Pack 1 return 0x00 (FALSE) when the named folder already exists.	Y	Content update.
2.2.2.2.7 CheapServerCount	55131 Added information about the relationship between this parameter and the ServerCount parameter.	Y	Content update.

Section	Tracking number (if applicable) and description	Major change (Y or N)	Change type
2.2.4.2.1 ReturnValue	55250 Added a table of possible values for ReturnValue. Added a product behavior note to define Exchange 2007 behavior regarding ecSearchFolderScopeViolation.	Y	New content added.

8 Index

A

Applicability 12

C

Capability negotiation 12
Change tracking 70
Client
  overview 37

E

Examples
  overview 50

F

Fields – vendor-extensible 12

G

Glossary 9

I

Implementer – security considerations 68
Index of security parameters 68
Informative references 11
Introduction 9

M

Messages
  overview 13
Messaging
  transport 13

N

Normative references 10

O

Overview (synopsis) 11

P

Parameters – security index 68
Preconditions 12
Prerequisites 12
Product behavior 69

R

References
  informative 11
  normative 10
Relationship to other protocols 11

S

Security
  implementer considerations 68
  overview 68
  parameter index 68
Server
  overview 41
Standards Assignments 12

T

Tracking changes 70
Transport 13

V

Vendor-extensible fields 12
Versioning 12