[MS-OXCFOLD]: Folder Object Protocol Specification

Intellectual Property Rights Notice for Open Specifications Documentation

- **Technical Documentation.** Microsoft publishes Open Specifications documentation for protocols, file formats, languages, standards as well as overviews of the interaction among each of these technologies.
- **Copyrights**. This documentation is covered by Microsoft copyrights. Regardless of any other terms that are contained in the terms of use for the Microsoft website that hosts this documentation, you may make copies of it in order to develop implementations of the technologies described in the Open Specifications and may distribute portions of it in your implementations using these technologies or your documentation as necessary to properly document the implementation. You may also distribute in your implementation, with or without modification, any schema, IDL's, or code samples that are included in the documentation. This permission also applies to any documents that are referenced in the Open Specifications.
- **No Trade Secrets**. Microsoft does not claim any trade secret rights in this documentation.
- **Patents**. Microsoft has patents that may cover your implementations of the technologies described in the Open Specifications. Neither this notice nor Microsoft's delivery of the documentation grants any licenses under those or any other Microsoft patents. However, a given Open Specification may be covered by Microsoft's Open Specification Promise (available here: http://www.microsoft.com/interop/osp) or the Community Promise (available here: http://www.microsoft.com/interop/cp/default.mspx). If you would prefer a written license, or if the technologies described in the Open Specifications are not covered by the Open Specifications Promise or Community Promise, as applicable, patent licenses are available by contacting iplg@microsoft.com.
- **Trademarks**. The names of companies and products contained in this documentation may be covered by trademarks or similar intellectual property rights. This notice does not grant any licenses under those rights.
- **Reservation of Rights**. All other rights are reserved, and this notice does not grant any rights other than specifically described above, whether by implication, estoppel, or otherwise.
- **Tools**. The Open Specifications do not require the use of Microsoft programming tools or programming environments in order for you to develop an implementation. If you have access to Microsoft programming tools and environments you are free to take advantage of them. Certain Open Specifications are intended for use in conjunction with publicly available standard specifications and

network programming art, and assumes that the reader either is familiar with the aforementioned material or has immediate access to it.

Revision Summary			
Author	Date	Version	Comments
Microsoft Corporation	April 4, 2008	0.1	Initial Availability.
Microsoft Corporation	April 25, 2008	0.2	Revised and updated property names and other technical content.
Microsoft Corporation	June 27, 2008	1.0	Initial Release.
Microsoft Corporation	August 6, 2008	1.01	Revised and edited technical content.
Microsoft Corporation	September 3, 2008	1.02	Revised and edited technical content.
Microsoft Corporation	December 3, 2008	1.03	Revised and edited technical content.
Microsoft Corporation	March 4, 2009	1.04	Revised and edited technical content.
Microsoft Corporation	April 10, 2009	2.0	Updated technical content and applicable product releases.

Table of Contents

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 the [MS-OXGLOS]:

active replica
contents table
EntryID
folder
folder associated information (FAI)
folder ID (FID)
Folder object
handle
little-endian
Logon object
mailbox
message
Message object
message ID (MID)
property
public folder
remote operation (ROP)
remote procedure call (RPC)
replica
ROP request buffer
ROP response buffer
rule

search folder
Server object
Server object handle table
soft delete
special folder
store
Unicode

The following data types are defined in [MS-OXCDATA]:

PtypBinary
PtypBoolean
PtypInteger32
PtypInteger64
PtypString
PtypTime

The following terms are specific to this document:

full text search: In text retrieval, a technique for searching a computer-stored document or database. In a **full text search**, the search engine examines all the words in every stored document as it tries to match search words supplied by the user.

hard delete: To permanently remove an item from the system. When a **message** or **folder** is deleted, a backup copy of that item can be kept by the server for a defined period of time. It is not possible for the messaging client to access or restore **hard deleted** items for any period of time.

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

[MS-OXCDATA] Microsoft Corporation, "Data Structures Protocol Specification", June 2008.

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

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

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

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

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

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

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

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

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

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

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

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

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

[MS-OXPROPS] Microsoft Corporation, "Exchange Server Protocols Master Property List Specification", June 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

None.

1.3 Protocol Overview

A **folder** is an object in a messaging **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 message store 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 message store.
- 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, and delete **folders**, to copy 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** 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 2.2.3.

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.

The complete syntax of the **RopOpenFolder** request and response buffers is specified in [MS-OXCROPS] section 2.2.3.1. 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 details 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 **FID** of the **folder** to be opened.

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 flags that can be set.

Name	Value	Description
OpenFolder	No flags set.	Indicates the opening of an existing folder.
OpenSoftDeleted	0x04	Indicates the opening of an

Name	Value	Description
		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 non-zero. If no rules are associated with this folder, the *flag* value is set to zero.<2>

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

2.2.1.2.3 IsGhosted

The *IsGhosted* parameter indicates whether the server hosts an **active replica** of the **folder**. If the server does not host an active replica of the folder, the server sets the **IsGhosted** field value to non-zero. Otherwise, this field value is set to zero, and the response buffer MUST contain the **ServerCount**, **CheapServerCount**, and **Servers** fields. The *IsGhosted* parameter is only present for folders in public **stores**.

For more details about ghosted folders, see **RopPublicFolderIsGhosted** <3> 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 non-zero.

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 non-zero.

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 non-zero.

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** <4> 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 2.2.3.3. This section specifies the syntax and semantics of various fields that are not fully specified in [MS-OXCROPS].

2.2.2.1 Request Parameters 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.

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 non-zero if **DisplayName** and **Comment** are formatted in **Unicode**. Otherwise, the *UseUnicodeStrings* parameter value is set to 0 (zero).

2.2.2.1.5 OpenExisting

If the *OpenExisting* parameter value is set to non-zero, 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 0 (zero).

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 **FolderDisplayName** in the request buffer already exists, the server sets the *IsExistingFolder* parameter value to non-zero. If the folder does not exist, the server sets the *IsExistingFolder* parameter value to zero. The following parameters are present only if the **FolderExistsFlag** is non-zero.

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 non-zero. If no rules are associated with this folder, the flag value is set to zero. The *HasRules* parameter is present only if *IsExistingFolder* parameter is non-zero and it is a **public folder store**.

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

2.2.2.2.5 IsGhosted

The *IsGhosted* parameter indicates whether the server hosts an **active replica** of the **folder**. If the server does not host an active replica of the folder, the server sets the *IsGhosted* parameter value to non-zero. Otherwise, this field value is set to 0 (zero), and the response buffer MUST contain the *ServerCount*, *CheapServerCount*, and *Servers* parameters. 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.

2.2.2.2.6 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 non-zero 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 field is only present if the *IsGhosted* parameter has a non-zero value.

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

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 non-zero value.

For more details about the *Servers* parameter, see **RopPublicFolderIsGhosted** in [MS-OXSTOR] 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 2.2.3.4. 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 flags that can be set.

Name	Value	Description
DEL_MESSAGES	0x01	Delete all the messages in the folder.
DEL_FOLDERS	0x04	Delete the subfolder and all its subfolders.
DELETE_HARD_DELETE	0x10	**Hard delete** the folder.

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 non-zero value.

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 non-zero. Otherwise, the *PartialCompletion* parameter value is 0 (zero).

2.2.4 RopDeletePublicFolderByName

The **RopDeleteFolderByName** operation deletes a subfolder by name. The subfolder name is relative to its parent folder or the root folder, and it does not include path information.

The complete syntax of the **RopDeletePublicFolderByName** request and response buffers is specified in [MS-OXCROPS] section 2.2.3.5. 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 DeleteFolderFlags

The *DeleteFolderFlags* parameter contains a bitmask of flags that control how to delete the folder. The following table lists the delete folder flags. These flags can be set.

Name	Value	Description
DEL_MESSAGES	0x01	Delete the **messages** of the folder.
DEL_FOLDERS	0x04	Delete subfolders.
DELETE_HARD_DELETE	0x10	**Hard delete** the **folder**.

2.2.4.1.3 NameSize

The *NameSize* parameter contains the size of the folder name, including the NULL character.

2.2.4.1.4 Name

The *Name* parameter contains a null-terminated string that is the name of the folder.

2.2.4.2 Response Parameter Overview

2.2.4.2.1 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.

2.2.4.2.2 PartialCompletion

If the operation failed for a subset of targets, the *PartialCompletion* parameter value is non-zero. Otherwise, the *PartialCompletion* parameter value is 0 (zero).

2.2.5 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.

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 2.14. 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.

The complete syntax of the **RopSetSearchCriteria** request and response buffers is specified in [MS-OXCROPS] section 2.2.3.6. 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

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 RestrictionDataSize

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

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

2.2.5.1.3 RestrictionData

The *RestrictionData* parameter contains a restriction. For more details about the structure of a restriction, see [MS-OXCDATA] section 2.14.

2.2.5.1.4 FolderIdCount

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

2.2.5.1.5 FolderIds

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

2.2.5.1.6 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 3.2.5.6.

The following table lists the flags that can be set.

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 containers that are specified in the folder list in the request buffer and all their child

Name	Value	Description
		folders. This flag cannot be set at the same time as the SHALLOW_SEARCH flag.
SHALLOW_SEARCH	0x00000008	The search only looks in the 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 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

Name	Value	Description
		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.

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.

2.2.6 **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 2.2.3.7. 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**

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

2.2.6.1.1 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.

2.2.6.1.2 UseUnicode

If results are required in **Unicode** format, the *UseUnicode* parameter is set to a non-zero value; otherwise, it is set to 0 (zero).

2.2.6.1.3 IncludeRestriction

If the restriction data is required in the response, the *IncludeRestriction* parameter is set to a non-zero value. Otherwise, it is set to 0 (zero).

2.2.6.1.4 IncludeFolders

If the **folders** list is required in the response, the *IncludeFolders* parameter is set to a non-zero value. Otherwise, it is set to 0 (zero).

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 RestrictionDataSize

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

2.2.6.2.3 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 2.14.

2.2.6.2.4 FolderIdCount

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

2.2.6.2.5 FolderIds

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

2.2.6.2.6 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 3.2.5.7.

The following table lists the flags that can be returned.

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 containers and all their child containers for matching entries. If this flag is not set, only the 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_STATIC	0x00010000	The search is static.
SEARCH_MAYBE_STATIC	0x00020000	The search is still being evaluated.

2.2.7 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, a **RopProgress**<5> response can be returned instead of the **RopMoveCopyMessages** response. 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 2.2.3.8. 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 MessageIdCount

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

2.2.7.1.4 MessageIds

The *MessageIds* parameter contains a list of **MIDs** to move or copy.

2.2.7.1.5 WantAsynchronous

The *WantAsynchronous* parameter value is set to non-zero 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 0 (zero). If the *WantAsynchronous* parameter value is non-zero, the **RopProgress** response can be returned instead of the **RopMoveCopyMessages** response. For more details about **RopProgress**, including usages, restrictions and notes, see [MS-OXCPRPT] section 2.2.22.

2.2.7.1.6 WantCopy

The *WantCopy* parameter is non-zero if this is a copy operation, or zero if this is a move operation.

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 non-zero. Otherwise, the *PartialCompletion* parameter value is 0 (zero).

2.2.8 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, a **RopProgress** response can be returned instead of the **RopMoveFolder** response. 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 2.2.3.9. 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 non-zero 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 0 (zero). If the *WantAsynchronous* parameter value is non-zero, the **RopProgress** response can be returned instead of the **RopMoveFolder**

response. For more details about **RopProgress**, including usages, restrictions, and notes, see [MS-OXCPRPT] section 2.2.22.

2.2.8.1.4 UseUnicode

If the *NewFolderName* parameter is formatted in **Unicode**, the *UseUnicode* parameter value is non-zero; otherwise, it is set to 0 (zero).

2.2.8.1.5 FolderId

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

2.2.8.1.6 NewFolderName

The *NewFolderName* parameter contains a null-terminated new **folder** name for the moved 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 non-zero. Otherwise, the *PartialCompletion* parameter value is 0 (zero).

2.2.9 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 on both **public folders** and private **mailboxes**. All **messages** in the source folder are duplicated on the new folder.

If the call is being processed asynchronously, a **RopProgress** response can be returned instead of the **RopCopyFolder** response. 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 2.2.3.10. 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 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.

2.2.9.1.2 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.

2.2.9.1.3 WantAsynchronous

The *WantAsynchronous* parameter value is set to non-zero 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 0 (zero). If the *WantAsynchronous* parameter value is non-zero, the **RopProgress** response can be returned instead of the **RopCopyFolder** response. For more details about **RopProgress**, including usages, restrictions, and notes, see [MS-OXCPRPT] section 2.2.22.

2.2.9.1.4 WantRecursive

The *WantRecursive* parameter is non-zero 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 0 (zero).

2.2.9.1.5 UseUnicode

If the *NewFolderName* parameter is formatted in **Unicode**, the *UseUnicode* parameter MUST be non-zero. Otherwise, it is set to 0 (zero).

2.2.9.1.6 FolderId

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

2.2.9.1.7 NewFolderName

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

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 non-zero. Otherwise, the *PartialCompletion* parameter value is 0 (zero).

2.2.10 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, a **RopProgress** response can be returned instead of the **RopEmptyFolder** response. 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 2.2.3.11. 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 non-zero 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 0 (zero). If the *WantAsynchronous* parameter value is non-zero, the **RopProgress** response can be returned instead of the **RopEmptyFolder** response. For more details about **RopProgress**, including usages, restrictions, and notes, see [MS-OXCPRPT] section 2.2.22.

2.2.10.1.3 WantDeleteAssociate

To delete all **messages**, including the **FAI** messages, the *WantDeleteAssociate* parameter value MUST be non-zero; otherwise, the *WantDeleteAssociate* parameter value is set to 0 (zero).

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 non-zero. Otherwise, the *PartialCompletion* parameter value is 0 (zero).

2.2.11 RopHardDeleteMessagesAndSubfolders

The **RopHardDeleteMessagesAndSubfolders** operation is used to **hard delete** all **messages** and subfolders from a **folder** without deleting the folder itself.<6>

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 a **RopProgress** response can be returned instead of the **RopHardDeleteMessagesAndSubfolders** response. 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 2.2.3.12. 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 non-zero 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 0 (zero). If the *WantAsynchronous* parameter value is non-zero, the **RopProgress** response can be returned instead of the **RopHardDeleteMessagesAndSubfolders** response. For more details about **RopProgress**, including usages, restrictions, and notes, see [MS-OXCPRPT] section 2.2.22.

2.2.11.1.3 WantDeleteAssociated

To delete all **messages**, including the **FAI** messages, the *WantDeleteAssociated* parameter value MUST be non-zero. Otherwise, the *WantDeleteAssociated* parameter value is set to 0 (zero).

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 non-zero. Otherwise, the *PartialCompletion* parameter value is 0 (zero).

2.2.12 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, a **RopProgress** response can be returned instead of the **RopDeleteMessages** response. 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 2.2.3.13. 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 non-zero 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 0 (zero). If the *WantAsynchronous* parameter value is non-zero, the **RopProgress** response can be returned instead of the **RopDeleteMessages** response. 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 0 (zero), the server does not generate a non-read receipt for the deleted **messages**. If the *NotifyNonRead* parameter value is non-zero, 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 details 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 non-zero. Otherwise, the *PartialCompletion* parameter value is 0 (zero).

2.2.13 RopHardDeleteMessages

The **RopHardDeleteMessages** operation **hard deletes** one or more **messages** that are listed in the request buffer<7>. 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, a **RopProgress** response can be returned instead of the **RopHardDeleteMessages** response. 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 2.2.3.14. 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 WantAsynchronous

The *WantAsynchronous* parameter value is set to non-zero 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 0 (zero). If the *WantAsynchronous* parameter value is non-zero, the **RopProgress** response can be returned instead of the **RopHardDeleteMessages** response. For more details about **RopProgress**, including usages, restrictions, and notes, see [MS-OXCPRPT] section 2.2.22.

2.2.13.1.3 NotifyNonRead

If the *NotifyNonRead* parameter is set to 0 (zero), the server does not generate a non-read receipt for the deleted **messages**. If the *NotifyNonRead* parameter value is set to non-zero, 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.

2.2.13.1.4 MessageIdCount

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

2.2.13.1.5 MessageIds

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

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 PartialCompletion

If the operation failed for a subset of targets, the *PartialCompletion* parameter value is non-zero. Otherwise, the *PartialCompletion* parameter value is 0 (zero).

2.2.14 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 2.2.3.15. 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 flags that can be set.

Name	Bitmask	Description
Depth	0x04	Fills the hierarchy table with containers from all levels. If this flag is not set, the hierarchy table contains only the container's immediate child 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 folder**s.

Name	Bitmask	Description
UseUnicode	0x40	Requests that the columns that contain string data be returned in **Unicode** format.
SuppressesNotifications	0x80	Suppresses notifications generated by this client's actions on this Table object.

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 hierarchy table. This field can be 0 (zero) instead of the actual count if the DeferredErrors flag is used.

2.2.15 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 2.2.3.16. This section specifies the syntax and semantics of various fields that are not fully specified in [MS-OXCROPS].

2.2.15.1 Request Parameter Overview

2.2.15.1.1 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.

2.2.15.1.2 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].

2.2.15.1.3 TableFlags

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

The following table lists the flags that can be set.

Name	Bitmask	Description
Associated	0x02	Requests an **FAI** table instead of a standard table. For more details 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	0x40	Requests that the columns that contain string data be returned in **Unicode** format.

2.2.15.2 Response Parameter Overview

2.2.15.2.1 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.

2.2.15.2.2 RowCount

The *RowCount* parameter contains the number of rows in the table. This field can be 0 (zero) 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 **Message 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 **FAI** entries in the **folder**. For more details 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 details 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 **PidTagNormalMessageSize** 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 non-zero; 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 user's **folder** permissions. For more details about folder permissions and valid **PidTagRights** values, see the values provided for the **PidTagMemberRights** property in [MS-OCXPERM] 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 2.2.15).

3.1.1.2 Contents Table

A **contents table** contains information about objects in a **message** 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 details about 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 2.2.16).

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 that the folder exists, and either opens the folder or creates it if it does not exist. Also, a user requires sufficient access rights to the folder for this operation to succeed.

To open an existing folder, an implementation sends the **RopOpenFolder**<8> 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. The *HandleIndex* that is returned by this **ROP** 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 Delete Folder by Name

RopDeleteFolderByName is analogous to **RopDeleteFolder**, except that the **folder** is specified by name instead of its **FID**.

3.1.4.5 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.

3.1.4.6 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**.

3.1.4.7 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.

3.1.4.8 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.

3.1.4.9 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.

3.1.4.10 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.

3.1.4.11 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.

3.1.4.12 Get 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.

3.1.4.13 Get Contents Table

RopGetContentsTable returns a Table object that contains information about **messages** in a message 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 non-zero 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 section 2.2.

RopOpenFolder will succeed only if a **folder** with the specified ID actually exists and the user 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 user does not have rights to the folder. OR The folder is soft deleted and the caller has not specified OpenSoftDeleted.
ecNotSupported	0x80040102	The object that this ROP was called on is not of type Folder or

Name	Value	Meaning
		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 **OpenIfExists**.

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 user 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 OpenIfExists flag was not specified.

Name	Value	Meaning
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 user 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 An attempt was made to delete a **special folder**. OR User does not have permissions to delete this folder.
ecNotFound	0x8004010F	Folder with the specified ID does not exist, or the user 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 RopDeletePublicFolderByName

RopDeletePublicFolderByName acts the same way as **RopDeleteFolder**, except that the **folder** is specified by name instead of by ID.

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 An attempt was made to delete a **special folder**. OR User does not have permissions to delete this folder.
ecNotFound	0x8004010F	Folder with the specified name does not exist or the user has no access to view that folder.
ecFolderHasChildren	0x80040609	The folder has subfolders and the DEL_FOLDERS flag was not specified.
ecFolderHasContents	0x8004060A	The folder has **messages** and the DEL_MESSAGES flag was not specified.
ecNotSupported	0x80040102	The object that this ROP was called on is not a **Folder object**.

3.2.5.5 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).

After search criteria are applied to a search folder, the user can query the contents of the search folder by using **RopGetContents** in the same way that the user 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 the uses of context indexing in server 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 for-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 **RopSearchCriteria** 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 **SetSearchCriteria** 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	0x00001121	The object is not a search folder.

Name	Value	Meaning
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**.

3.2.5.6 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	0x00001121	The object is not a search folder.
ecNotSupported	0x80040102	The object that this ROP was called on is not a **Folder object**.

3.2.5.7 RopMoveCopyMessages

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

If the client requests asynchronous execution, the server can execute 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**.

3.2.5.8 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 can execute 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 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 can execute 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**.

3.2.5.10 RopEmptyFolder

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

If the WantDeleteEverything flag is specified, 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 can execute 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 non-zero 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**.

3.2.5.11 RopHardDeleteMessagesAndSubfolders

RopHardDeleteMessagesAndSubfolders behaves in the same way as **RopEmptyFolder**, except that **messages** and subfolders that are removed by this **ROP** is **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**.

3.2.5.12 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 can execute 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**.

3.2.5.13 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 can execute 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

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

3.2.5.14 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 SuppressNotifications flag is specified, actions from this client do not trigger events on this table.

The **RowCount**<9> 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.

3.2.5.15 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 **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 details 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 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 underlined

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 **HandleIndex** values, the example text describes the **Server object** that is referenced by the **handle** index. For more details 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 2.2.3.

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**, **LogonIndex**, **HandleIndex**, and **FolderHandleIndex** fields of the **RopCreateFolder** format, as specified in section 2.2.3.

```
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**, **UseUnicodeString**, **OpenExisting**, and **HasLongTermId** fields of the **RopCreateFolder** format, as specified in section 2.2.3. These fields affect how the operation is carried out.

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

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

UseUnicodeString: "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.

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

The next 10 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 2.2.3.2:

```
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 2.2.4.

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.

DeleteFolderFlag: "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 2.2.13. In this example, a **folder** contains two **messages**, **MessageId** 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 2.2.13.

```
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 caller 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 2.2.8. 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 2.2.8.

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

RopId: "0x33" (**RopMoveCopyMessages**)

LogonIndex: "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". **MessageId** 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 2.2.9. 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 2.2.9.

```
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 2.2.10. In this example, a **folder**, specified by its **FolderId**, 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 2.2.10.

```
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.

DestFolderHandleIndex: "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 01 00 00 00 00 00
```

RopId: "0x36" (**RopCopyFolder**)

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.

4.7 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 2.2.15. 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 **TableFlags** values, see section 2.2.15.1.3.

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 2.2.6. The **search folder** is referred to by the **HandleIndex** 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 2.14.

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**, **LogonIndex**, and **HandleIndex** fields of the **RopCopyFolder** format, as specified in section 2.2.6.

```
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 **SRestriction** 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 "PM.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.

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 2.2.7. The **search folder** is referred to by the **HandleIndex** 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 2.14.

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<10>: "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 2.2.7.2:

```
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 **SRestrictions** that defines 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 "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 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 five bytes of the server response buffer contain the **LogonIndex** and **SearchStateFlag** fields.

```
0133: 00 01 00 00 00
```

LogonIndex: "0x00". This is the **LogonIndex** used in a previous **RopLogon** call.

SearchStateFlags: "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: Office/Exchange Behavior

The information in this specification is applicable to the following versions of Office/Exchange:

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

<1> Section 2.2: Exchange 2010 Beta 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:

RopAbortSubmit

RopBackOff

RopCollapseRow

RopCopyToStream

RopCreateBookmark

RopDeletePropertiesNoReplicate

RopExpandRow

RopFastTransferSourceCopyTo

RopFreeBookmark

RopGerPerUserGuid

RopGetCollapseState

RopGetOwningServers

RopGetPerUserLongTermIds

RopGetReceiveFolderTable

RopGetStatus

RopGetStoreState

RopHardDeleteMessages

RopHardDeleteMessagesAndSubfolders

RopLockRegionStream

RopPending

RopPublicFolderIsGhosted

RopQueryColumnsAll

RopQueryNamedProperties

RopReadPerUserInformation

RopRegisterSynchronizationNotifications

RopSeekRowBookmark

RopSeekRowFractional

RopSeekStream

RopSetCollapseState

RopSetPropertiesNoReplicate

RopSetReceiveFolder

RopSetSynchronizationNotificationGuid

RopSynchronizationOpenAdvisor

RopUnlockRegionStream

RopUpdateDeferredActionMessages

RopWritePerUserInformation

<2> Section 2.2.1.2.2: Exchange 2003 SP2 and Exchange 2007 SP1 return a value of FALSE for the *HasRules* parameter, even when there are rules on the Inbox.

<3> Section 2.2.1.2.3: Exchange 2010 Beta does not support this ROP when client connection services are deployed on an Exchange server that does not also have a mailbox store installed.

<4> Exchange 2010 Beta supports public folder referrals, but does not support public folders when client connection services are deployed on an Exchange server that does not also have a mailbox store installed.

<5> Exchange 2010 Beta can output unexpected results for **RopProgress** when client connection services are deployed on an Exchange server that does not also have a mailbox store installed.

<6> Section 2.2.11: Exchange 2010 Beta does not support this ROP when client connection services are deployed on an Exchange server that does not also have a mailbox store installed.

<7> Section 2.2.13: Exchange 2010 Beta does not support this ROP when client connection services are deployed on an Exchange server that does not also have a mailbox store installed.

<8> Exchange 2010 Beta can output unexpected results when using **RopOpenFolder** when client connection services are deployed on an Exchange server that does not also have a mailbox store installed.

<9> Exchange 2010 Beta can output unexpected results when the *RowCount* parameter is used and client connection services are deployed on an Exchange server that does not also have a mailbox store installed.

<10> Exchange 2010 Beta does not support the **UseUnicode** flag not being set when client connection services are deployed on an Exchange server that does not also have a mailbox store installed.

Index