[MS-OXCPRPT]: Property and Stream Object Protocol Specification

Intellectual Property Rights Notice for Protocol Documentation

Tools. This protocol documentation is intended for use in conjunction with publicly available standard specifications and networking programming art, and assumes that the reader is either familiar with the aforementioned material or has immediate access to it. A protocol specification does not require the use of Microsoft programming tools or programming environments in order for a Licensee to develop an implementation. Licensees who have access to Microsoft programming tools and environments are free to take advantage of them.

Revision Summary			
Author	Date	Version	Comments
Microsoft Corporation	April 4, 2008	0.1	Initial Availability

Table of Contents

1 Introduction

The Property and Stream Object protocol specifies how to set, get, and delete data from a property set.

1.1 Glossary

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

Attachment objectattachment table
code page
global identifier (GID)
handle
little-endian
messaging object
multiple-byte character set (MBCS)
named property
property
property ID
property name
property tag
recipient table
remote operation (ROP)
ROP request buffer
ROP response buffer
stream
Unicode

The following data types are defined in [MS-DTYP]

BYTE
Boolean
String
ULONG
WORD

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", April 2008.

[MS-OXCFOLD] Microsoft Corporation, "Folder Object Protocol Specification", April 2008.

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

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

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

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

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

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

1.2.2 Informative References

None.

1.3 Protocol Overview (Synopsis)

The Property and Stream Object protocol specifies how to set, get and delete properties associated with a messaging object. In order to get or set properties associated with a messaging object, you first need to acquire a handle to the object. Methods to acquire handles to the objects is dependent on the object type and are specified in detail for **Messages** and **Attachments** in [MS-OXCMSG], for **Folders** in [MS-OXCFOLD], and for **Logons** in [MS-OXCSTOR].

1.3.1 Setting, Getting, and Deleting Properties

Properties are used to store data about messaging objects. Properties can store data of various types, such as **Boolean**, **String**, **Int32**, and **Binary** data (See [MS-OXCPROPS for a complete list]). A property is identified by property tag which the client uses to set, get, or delete the properties. The messaging client adds or changes data on these properties using **RopSetProperties**. This data is retrieved using **RopGetPropertiesSpecific** or **RopGetPropertiesAll**. Data is removed using **RopDeleteProperties**. The messaging client can get a list of properties which have been set on an object using **RopGetPropertiesList**.

1.3.2 Named Properties Operations

In addition to static **Property ID**s, clients can create additional properties using **Named Properties**. Named Properties provide a mechanism for clients to map their own Property IDs with **Property Tag**s understood by the server. Using **RopGetPropertyIdsFromNames**

a messaging client can create new mappings from **Property Name** to Property ID. In addition **RopGetPropertyIdsFromNames** can be used to retrieve the Property Id from an existing mapping.**RopQueryNamedProperties** can be used to retrieve a whole list of Property Names. The client can also use **RopGetNamesFromPropertyIds** to retrieve the Property Name from a Property ID.

1.3.3 Copying Properties

Properties can be copied from one object to another using **RopCopyProperties** to copy just a select few properties, or **RopCopyTo** to copy all, or almost all properties.

1.3.4 Using Streams to Read and Write Properties

Some properties on messages and attachments can be opened as a stream. To use a stream the client can use the **RopOpenStream ROP** to open a stream. With an open stream the client can use **RopReadSteam** or **RopWriteStream** to read from or write to the stream. And finally **RopCommitStream** to save the stream back to the property.

1.3.5 Saving

There are two different transaction models for property objects. Changes to properties on **Folder** objects and **Logon** objects are saved immediately, whereas changes to **Message** properties are not saved until the client sends a **RopSaveMessage**, and **Attachment** objects are not saved until the client sends a **RopSaveChangesAttachment**. See [MS-OXCMSG] for more details on how **Messages** are saved.

1.4 Relationship to Other Protocols

The Property and Stream Object protocol specification relies on understanding how ROPs are transmitted to the server using the underlying Remote Procedure Call (RPC) transport (see [MS-OXCROPS] and [MS-OXCRPC]).

1.5 Prerequisites/Preconditions

The Property and Stream Object protocol specification assumes the messaging client has previously logged on to the messaging server and has acquired a handle to the property object on which it is going to operate. Methods to open the object and acquire a handle is dependent on the object type and are specified in detail for **Messages** and **Attachments** in [MS-OXCMSG], for **Folders** in [MS-OXCFOLD], and for **Logons** in [MS-OXCSTOR].

1.6 Applicability Statement

The Property and Stream Object protocol is used to query or modify properties associated with messaging objects on the server.

1.7 Versioning and Capability Negotiation

None.

1.8 Vendor-Extensible Fields

1.8.1 Named Properties

Named Properties allow vendors or individual clients to use additional **Property IDs** without having to worry about reusing properties that other vendors are using. Named Properties provide a persistent mapping from **Property Names** to Property IDs. The client can create new named properties, and get the property ID for existing property names using **RopGetPropertyIdsFromNames.**

1.9 Standards Assignments

None.

2 Messages

Unless otherwise specified, sizes in this section are expressed in BYTEs.

2.1 Transport

The **ROP Request Buffers** and **ROP Response Buffers** specified by this protocol are sent to and are received from the server using the underlying Remote Procedure Call (RPC) transport as specified in [MS-OXCROPS] and [MS-OXCRPC].

2.2 Message Semantics

The following sections specify the usage of **ROP** requests and responses that are specific to the Property and Stream Object protocol.

2.2.1 Common Object Properties

When a property is referred to as "read-only for the client", the server MUST return an error and ignore any request to change the value of that property.

2.2.1.1 PidTagAccess

Type: PtypInteger32.

Indicates the operations available to the client for the object. MUST be a bitwise-OR of zero or more values from the following table. This property is read-only for the client.

Value	Meaning
0x00000001	Modify
0x00000002	Read
0x00000004	Delete
0x00000008	Create Hierarchy Table
0x00000010	Create Contents Table
0x00000020	Create Associated Contents Table

2.2.1.2 PidTagAccessLevel

Type: PtypInteger32.

Indicates the client's access level to the object. This property is read-only for the client. MUST be one of the following values:

Value	Meaning
0x00000000	Read-Only
0x00000001	Modify

2.2.1.3 PidTagChangeKey

Type: PtypBinary.

Contains a global identifier (GID) indicating the last change to the object. See **PidTagChangeKey** in [MS-OXCFXICS].

2.2.1.4 PidTagCreationTime

Type: PtypTime, in UTC.

Contains the time the object was created.

2.2.1.5 PidTagLastModifierName

Type: PtypString.

Contains the name of the last messaging user to modify the object.

2.2.1.6 PidTagLastModificationTime

Type: PtypTime, in UTC.

Contains the time of the last modification to the object.

2.2.1.7 PidTagObjectType

Type: PtypInteger32.

Indicates the type of server object. This property is read-only for the client. MUST be one of the following values:

Value	Meaning
0x00000001	Store Object
0x00000002	Address Book Object
0x00000003	Folder Object
0x00000004	Address Book Container Object
0x00000005	Message Object
0x00000006	Messaging User Object
0x00000007	Attachment Object
0x00000008	Distribution List Object

0x0000000A	Status Object

2.2.1.8 PidTagRecordKey

Type: PtypBinary.

Contains a unique binary-comparable identifier for a specific object. Whenever a copy of this object is created the server generates a new value for the new object. This property is read-only for the client.

2.2.1.9 PidTagSearchKey

Type: PtypBinary.

Contains a unique binary-comparable key that identifies an object for a search. Whenever a copy of this object is created this value is also copied from the original object.

2.2.2 RopGetPropertiesSpecific

RopGetPropertiesSpecific queries for and returns the values of properties specified in the property array field. The property values MUST be returned in the same order as they were requested.   Objects that are supported for this operation are: **Message**, **Folder**, **Attachment** and **Logon.**The syntax of this ROP is fully specified in [MS-OXCROPS].

2.2.2.1 Request Parameter Overview

See [MS-OXCROPS] for a detailed view of the format of the **RopGetPropertiesSpecific** ROP. The values of the items in the ROP are described in the following sections.

2.2.2.1.1 PropertySizeLimit

Maximum size allowed for a property value. Properties larger than this *PropertySizeLimit* MUST get a **NotEnoughMemory** [MS-OXPROPS] error returned for their value in the response. If this value is zero, the property values are only limited by the size of the response buffer. If this value is non-zero, the property value are limited by the size of the response buffer and the value of *PropertySizeLimit.*

2.2.2.1.2 WantUnicode

If non-zero, indicates that string properties which are requested with **PtypUnspecified** [MS-OXCDATA] are encoded by the server using the Unicode format. If *WantUnicode* is zero, the server MUST return string properties which are requested with **PtypUnspecified** [MS-OXCDATA] as the property type using a **MBCS Character Set**.

Properties requested with a specific string type MUST be returned using that type.

For properties on **Message** objects the code page used for strings in MBCS format MUST be the code page set on the message (See [MS-OXCMSG for details) when it was opened if any, otherwise the code page of the **Logon**.

For properties on **Attachment** objects the code page used for strings in MBCS format MUST be the code page set on the parent message when it was opened if any otherwise the code page of the **Logon**.

All other objects the code page used for strings in MBCS format MUST be the code page of the **Logon**.

2.2.2.1.3 PropertyTagCount

Count of the property tags in the *PropertyTags* field.

2.2.2.1.4 PropertyTags

Array of property tags.This is a list of property tags for which the client is requesting the value.The server MUST order properties in the *PropertyValues* element of the response buffer in the same order in which properties are specified in this *PropertyTags*. The buffer format of a property tag is specified in [MS-OXCDATA].

2.2.2.2 Response Parameter Overview

2.2.2.2.1 RowData

The *PropertyArray* field is an array of **PropertyRow** [MS-OXCDATA] elements. The number of items in the array is specified in the *PropertyCount* field of the request buffer. Properties larger than the requested *PropertySizeLimit* the server MUST return a **NotEnoughMemory** [MS-OXPROPS] error for their value in the response.

2.2.3 RopGetPropertiesAll

RopGetPropertiesAll queries for and returns all of the property tags and values that have been set. The server MUST return all properties necessary for the client to create an equivalent duplicate by copying only these properties, without considering its **Attachment Table** and **Recipient Table** that might be on the object. Objects that are supported for this operation are: **Message**, **Folder**, **Attachment** and **Logon.** The syntax of this ROP is fully specified in [MS-OXCROPS].

2.2.3.1 Request Parameter Overview

2.2.3.1.1 PropertySizeLimit

Maximum size allowed for a property value. Properties larger than this *PropertySizeLimit* MUST get a **NotEnoughMemory** [MS-OXPROPS] error returned for their value in the response. If this value is zero, the property values are only limited by the size of the response buffer. If this value is non-zero, the property value are limited by the size of the response buffer and the value of *PropertySizeLimit.*

2.2.3.1.2 WantUnicode

If non-zero, indicates that string properties which are requested with **PtypUnspecified** [MS-OXCDATA] are encoded by the server using the Unicode format. If *WantUnicode* is zero, the server MUST return string properties which are requested with **PtypUnspecified** [MS-OXCDATA] as the property type using a **MBCS CodePag**e.

Properties requested with a specific string type MUST be returned using that type.

For properties on **Message** objects the code page used for strings in MBCS format MUST be the code page set on the message when it was opened if any, otherwise the code page of the **Logon**.

For properties on **Attachment** objects the code page used for strings in MBCS format MUST be the code page set on the parent message when it was opened if any otherwise the code page of the **Logon**.

All other objects the code page used for strings in MBCS format MUST be the code page of the **Logon**.

2.2.3.2 Response Parameter Overview

2.2.3.2.1 PropertyValueCount

The value of the *PropertyTagCount* field MUST be the number of elements in *PropertyValues*.

2.2.3.2.2 PropertyValues

The *PropertyValues* field is a list of all property types and property values on the queried object. If a value was too big to fit in the response or if the property value is larger than *PropertySizeLimit* the type MUST be **PtypErrorCode** [MS-OXCDATA] with a value of **NotEnoughMemory** [MS-OXPROPS].

2.2.4 **RopGetPropertiesList**

RopGetPropertiesList queries for and returns all of the property tags that have been set.. The server MUST return all properties necessary for the client to create an equivalent duplicate by copying only these properties. without considering its **Attachment Table** and **Recipient Table** that might be on the object. Objects that are supported for this operation are: **Message**, **Folder**, **Attachment** and **Logon.** The syntax of this ROP is fully specified in [MS-OXCROPS].

2.2.4.1 Response Parameter Overview

2.2.4.1.1 PropertyTagCount

The value of the *PropertyTagCount* field is the number of elements in *PropertyTags*.

2.2.4.1.2 PropertyTags

If the *ReturnValue* is zero, the *PropertyTags* field contains a list of property tags for each property currently set on the object.

2.2.5   RopSetProperties

RopSetProperties updates values of properties specified in the property array field. Objects that are supported for this operation are: **Message**, **Folder**, **Attachment** and **Logon.** The syntax of this ROP is fully specified in [MS-OXCROPS].

2.2.5.1 Request Parameter Overview

2.2.5.1.1   PropertyValueSize

The value of the *PropertyValueSize* field is the number of bytes in *PropertyValueCount* plus the number of bytes in *PropertyValues*.

2.2.5.1.2   PropertyValueCount

The value of the *PropertyValueCount* field is the number of elements in *PropertyValues*.

2.2.5.1.3   PropertyValues

The *PropertyValues* field contains a list of **TaggedPropertyValue** elements as specified in [MS-OXCDATA].

2.2.5.2 Response Parameter Overview

2.2.5.2.1   PropertyProblemCount

The value of *PropertyProblemCount* is the number of elements in *PropertyProblems*.

2.2.5.2.2   PropertyProblems

The value of *PropertyProblems* is a list of any properties that were unable to be set, and the reason the property was unable to be set. See **PropertyProblem** in [MS-OXCDATA] for the format of this field.

2.2.6   RopSetPropertiesNoReplicate

RopSetPropertiesNoReplicate has the same parameters and works the same way as **RopSetProperties**, with the exception that when used to set properties on a **Folder** object, the set properties will not undergo folder replication (See [MS-OCXSTOR] for details on folder replication) On all other objects, **RopSetPropertiesNoReplicate** works exactly the same as **RopSetProperties**.

2.2.7   RopDeleteProperties

RopDeleteProperties removes the values of properties specified in the *PropertyValues* field from the base object. Objects that are supported for this operation are: **Message**, **Folder**, **Attachment** and **Logon.** The syntax of this ROP is fully specified in [MS-OXCROPS].

2.2.7.1 Request Parameter Overview

2.2.7.1.1 PropertyTagCount

The value of *PropertyTagCount* is the number of elements in *PropertyTags*.

2.2.7.1.2 PropertyTags

PropertyTags contains a list of property tags for the properties that the client is deleting. See **PropertyTag** in [MS-OXCDATA].

2.2.7.2 Response Parameter Overview

2.2.7.2.1 PropertyProblemCount

The value of *PropertyProblemCount* is the number of elements in *PropertyProblems*.

2.2.7.2.2 PropertyProblems

The *PropertyProblems* field contains a list of which properties the server failed to delete, and the reason the property was unable to be deleted. See **PropertyProblem** in [MS-OXCDATA] for the format of this field.

2.2.8 RopDeletePropertiesNoReplicate

RopDeletePropertiesNoReplicate has the same parameters and works the same way as **RopDeleteProperties**, with the exception that when used to delete properties from a **Folder** object, the deleted properties will not undergo folder replication.(See [MS-OCXSTOR] for details on folder replication) On all other objects, **RopDeletePropertiesNoReplicate** works exactly the same as **RopDeleteProperties.**

2.2.9 RopQueryNamedProperties

RopQueryNamedProperties queries an object for all the named properties and their IDs. Objects that are supported for this operation are: **Message** and **Attachment**. The syntax of this ROP is fully specified in [MS-OXCROPS].

2.2.9.1 Request Parameter Overview

2.2.9.1.1 QueryFlags

QueryFlags is a BYTE bit field. Any bits not specified below SHOULD be ignored by the server.

Name	Value	Description
NoStrings	0x01	Named Properties with a **Kind** [MS-OXCDATA] of 0x1 MUST NOT be included in the response.

NoIds	0x02	Named Properties with a **Kind** [MS-OXCDATA] of 0x0 MUST NOT be included in the response.

2.2.9.1.2 HasGUID

If the *HasGUID* field is non-zero then the *PropertyGUID* field MUST be included in the request. If no *PropertyGUID* is specified, then properties from any GUID MUST be returned in the results. If the *HasGUID* field is zero then the *PropertyGUID* field MUST NOT be present.

2.2.9.1.3 PropertyGUID

Only named properties with GUID's matching *PropertyGUID* MUST be returned in a successful response.

2.2.9.2 Response Parameter Overview

2.2.9.2.1 IdCount

The value of *IdCount* is the number of elements in *PropertyIds* and in *NameIdList* They MUST have the same number of elements.

2.2.9.2.2 PropertyIds

The *PropertIds* field contains a list of **Property Identifiers** (See [MS-OXCDATA]), one for each of the named properties which match the *PropertyGUID* requested.

2.2.9.2.3 PropertyNames

The *PropertyNames* field contains a list of Property Names in the format specified in [MS-OXCROPS]. This field MUST contain *IdCount* entries. The entries in this list MUST match the order of the entries in *PropertyIds* to form Property Id to Name pairs for each named property matching the *PropertyGUID* specified in the request.

2.2.10 RopCopyProperties

RopCopyProperties copies or moves one or more properties from one object to another. It can be used for replying to and forwarding messages, in which only some of the properties from the original message travel with the reply or forwarded copy.This ROP is valid on objects of type **Folder, Attachment** and **Message.** Also, the source and destination object MUST be of the same type. The syntax of this ROP is fully specified in [MS-OXCROPS].

2.2.10.1 Request Parameter Overview

2.2.10.1.1 WantAsynchronous

The *WantAsynchronous* field value is non-zero to request this ROP's operation to be performed asynchronously. For the operation to be performed synchronously the *WantAsynchronous* field value MUST be zero. If the *WantAsynchronous* field value is non-zero the **RopProgress** response MAY be returned instead of the **RopMoveCopyMessages** response. See **RopProgress** for more details.

2.2.10.1.2 CopyFlags

CopyFlags is a BYTE bit field. Any bits not specified below MUST be ignored by the server.

Name	Value	Description
Move	0x01	If set, makes the call a move operation rather than a copy operation.
NoOverwrite	0x02	If set, any properties being set by **RopCopyProperties** that already have a value on the destination object will not be overwritten; otherwise, they are overwritten.

2.2.10.1.3 PropertyTagCount

The value of *PropertyTagCount* is the number of elements in *PropertyTags*.

2.2.10.1.4 PropertyTags

The *PropertyTags* field contains a list of **Property Tags** [MS-OXCDATA] to be copied or moved.

2.2.10.2 Response Parameter Overview

2.2.10.2.1 PropertyProblemCount

The value of *PropertyProblemCount* is the number of elements in *PropertyProblems*.

2.2.10.2.2 PropertyProblems

The *PropertyProblems* field contains a list of which properties the server failed to copy, and the reason the property was unable to be copied. See **PropertyProblem** in [MS-OXCDATA] for the format of this field.

2.2.11 RopCopyTo

The **RopCopyTo** ROP is used to copy or move all but a specified few properties from a source object to a destination object. The ROP is valid on **Message, Attachment** and **Folder** objects. The client can specify whether to copy the sub-objects in the source object as a flag in request buffer.

If any of the copied or moved properties already exist in the destination object, the existing properties are overwritten by the new properties, unless *CopyFlags* in the request buffer

specifies the 0x02 bit. Existing information in the destination object that is not overwritten is left untouched.

The syntax of this ROP is fully specified in [MS-OXCROPS].

2.2.11.1 Request Parameter Overview

2.2.11.1.1 WantAsynchronous

The *WantAsynchronous* field value MUST be non-zero to request this ROP's operation to be performed asynchronously. For the operation to be performed synchronously the *WantAsynchronous* field value MUST be zero. If the *WantAsynchronous* field value is non-zero the **RopProgress** Response MAY be returned instead of the **RopCopyTo** Response. See **RopProgress** for more details.

2.2.11.1.2 WantSubObjects

If *WantSubObjects* is non-zero then sub-objects MUST also be copied. Otherwise they are not.

2.2.11.1.3 CopyFlags

CopyFlags is a bit field of the following values.

Name	Value	Description
Move	0x01	If set, makes the call a move operation rather than a copy operation. This will delete the moved properties from the source object.
NoOverwrite	0x02	If set, any properties being set by **RopCopyTo** that already have a value on the destination object will not be overwritten; otherwise, they are overwritten.

2.2.11.1.4 ExcludedTagCount

The value of *ExcludedTagCount* is the number of elements in *ExcludedTags*.

2.2.11.1.5 ExcludedTags

The value of *ExcludedTags* is a list of Property Tags (As specified in [MS-OXCDATA]). This is a list of properties that MUST NOT be copied or moved as part of this operation.

2.2.11.2 Response Parameter Overview

2.2.11.2.1 PropertyProblemCount

The value of *PropertyProblemCount* is the number of elements in *PropertyProblems*.

2.2.11.2.2 PropertyProblems

The *PropertyProblems* field contains a list of which properties the server failed to copy, and the reason the property was unable to be copied. See **PropertyProblem** in [MS-OXCDATA] for the format of this field.

2.2.12 RopGetPropertyIdsFromNames

The **RopGetPropertyIdsFromNames** remote operation (ROP) is used to map abstract, client-defined property names into concrete 16-bit property IDs (15 of which are significant). A fully formed property tag is built from the 16-bit property ID and a 16-bit type code. All operations with properties, (setting, getting, deleting) on messaging objects MUST be done with the concrete 32-bit property tag. In case the predefined set of Property Tags does not cover a semantic need for a new client feature, the client is able to register new property IDs using this remote operation. The client provides a namespace identifier, a property identifier in either integer or string form and the server returns a property ID to use when the client wishes to make use of that property. String-form names are case-sensitive, except for names in the *PS-INTERNET-HEADERS* namespace, which are always coerced to lower case.

Because there are only 32,767 available property IDs (fifteen significant bits), the server MAY impose a lower limit on the total number of mappings it will allow before returning errors to the client.

This ROP is valid on **Message, Attachment, Folder** and **Logon** objects. When issued against a **Logon**, the messaging client MAY request to map zero names to IDs. In that event, the server MUST return a comprehensive list of all registered named property IDs.

2.2.12.1 Request Parameter Overview

2.2.12.1.1 Flags

Name	Value	Description
Create	0x00000002	If set, indicates that the server MUST create new entries if any name parameters are not found in the existing mapping set.

2.2.12.1.2 PropertyNameCount

The number of **PropertyName** structures that follow. MAY be zero. Zero indicates the messaging client is querying the complete list of registered named property tags.

2.2.12.1.3 PropertyNames

An array of **PropertyName** [MS-OXCDATA] structures, the length of which MUST be *Count* elements.

2.2.12.1.3.1 PropertyIdCount

The number of 16-bit property IDs that follow. MUST be equal to the *Count* parameter unless the *Count* parameter was zero. In that case, MUST be equal to the total number of registered named properties.

2.2.12.1.3.2 PropertyIds

An array of 16-bit property IDs. Total item length MUST be *Count*. For names that could not be mapped, the associated value MUST be zero. In this event, the *Return Code* MUST be equal to ecWarnWithErrors (0x00040380). The order of IDs in this array MUST match the order of the names specified in the *PropertyNames* parameter.

For any entries in the *PropertyNames* array with a *NamespaceGUID* equal to *PS-MAPI* [MS-OXPROPS], the resultant entry in the *PropertyIds* MUST be equal to the *IntegerID* value from the *PropertyNames* entry. For other namespaces, the resultant entry MUST have the most significant bit set (0x8000) to indicate the property ID is a named property.

Reasons a name couldn't be mapped include:

- Use of the PS-MAPI [MS-OXPROPS] namespace and not specifying 0x00 for **Kind** [MS-OXCDATA]
- The name wasn't found in the mapping table and the *Create* flag wasn't set in the *Flags* parameter
- The user does not have permission to register new named properties
- The user has reached an artificial quota of named properties imposed by the server
- The user has reached the hard limit of 32,767 registered named properties.

2.2.13 **RopGetNamesFromPropertyIds**

The **RopGetNamesFromPropertyIds** remote operation (ROP) is used to map concrete property IDs to abstract, client-defined property names. A fully formed property tag is built from the 16-bit property ID and a 16-bit type code. All operations with properties, (setting, getting, deleting) on messaging objects MUST be done with the concrete 32-bit property tag. In case the predefined set of property tags does not cover a semantic need for a new client feature, the client is able to register new property IDs using **RopGetPropertyIdsFromNames**. The client provides a namespace identifier, a property identifier in either integer or string form and the server returns a property ID to use when the client wishes to make use of that property. This remote operation is used to recover the abstract name from previously registered named property IDs.

This ROP is valid on **Message, Attachment, Folder** and **Logon** objects. Named property IDs are identified by having their most significant bit set (0x8000). The client MAY request to map non-named property IDs (IDs with the most significant bit unset) into names. The abstract name for such IDs is comprised of the *PS-MAPI* [MS-OXPROPS] namespace GUID, a **Kind** [MS-OXCDATA] value equal to 0x00, and a **LID** [MS-OCXDATA] value equal to the property ID given.

The PS-MAPI [MS-OXPROPS] namespace used for mapping non-named property IDs into names.

2.2.13.1 Request Parameter Overview

2.2.13.1.1 PropertyIdCount

The number of 16-bit property IDs in *PropertyIds*.

2.2.13.1.2 PropertyIds

An array of 16-bit property IDs to map into abstract names. The length MUST be equal to *PropertyIdCount* elements. Typically, the array contains only values greater than 0x8000, indicating that they are in fact named property IDs. The client MAY pass property ID values less than 0x8000. These values will be mapped into names in the *PS-MAPI* [MS-OXPROPS] namespace.

2.2.13.2 Response Parameter Overview

If the *Return Code* value is **Success** [MS-OXCDATA] or **ErrorsReturned** [MS-OXCDATA], the following fields are present.

2.2.13.2.1 PropertyNameCount

The number of elements in *PropertyNames*. Value MUST be equal to the *PropertyIdCount* parameter.

2.2.13.2.2 PropertyNames

An array of **PropertyName** [MS-OXCDATA] structures.

2.2.14 RopOpenStream

RopOpenStream is used to open a property to perform a streaming operation. This ROP enables the client to perform various streaming operations on the specified property tag.

This ROP is valid on **Folder, Attachment,** and **Message** objects.

Only single valued Binary(**PtypBinary** [MS-OXCDATA]), Object (**PtypObject** [MS-OXCDATA]), String (**PtypString8** [MS-OXCDATA]), and **PtypString** [MS-OXCDATA]) type properties are supported. **Folders** only support Binary (**PtypBinary** [MS-OXCDATA]) type properties.

The client MUST use **RopRelease** to release the stream once it is done the stream object.

The syntax of this ROP is fully specified in [MS-OXCROPS].

2.2.14.1 Request Parameter Overview

2.2.14.1.1 PropertyTag

The *PropertyTag* field specifies which property the client is opening. It is in the format of a Property Tag as specified in [MS-OXCDATA].

2.2.14.1.2 OpenModeFlags

The value of *OpenModeFlags* MUST be one of the following values, which MAY also be bitwise ANDed with 0x04 to indicate that the stream MUST be seeked to the end of the current data.

Name	Value	Description
ReadOnly	0x00	Open stream for read-only access.
ReadWrite	0x01	Open stream for read/write access.
Create	0x02	Opens new stream, this will delete the current property value and open stream for read/write access. This is required to open a property that has not been set.
BestAccess	0x03	If the object this ROP is acting on was opened with **ReadWrite** access, then the stream MUST be opened with **ReadWrite** access. Otherwise, the stream MUST be opened with **ReadOnly** access.

2.2.14.2 Response Parameter Overview

2.2.14.2.1 StreamSize

StreamSize MUST be the current number of BYTES in the stream.

2.2.15 RopReadStream

RopReadStream is used to read the stream of bytes from a stream object.This ROP is valid only on **Stream** objects. The syntax of this ROP is fully specified in [MS-OXCROPS].

2.2.15.1 Request Parameter Overview

2.2.15.1.1 ByteCount

The value of *ByteCount* SHOULD be the size of the data buffer that will be sent back in the response. If the value is 0xBABE then the server will determine what buffer size to use in the response.

2.2.15.1.2 MaximumByteCount

This field MUST be present if and only if the value of *ByteCount* is 0xBABE. The value of *MaximumByteCount* specifies the maximum buffer size that the server can return in the response.

2.2.15.2 Response Parameter Overview

2.2.15.2.1 DataSize

The value of *DataSize* is the number of bytes in *Data*.

If *ByteCount* was specified as 0xBABE, the value of *DataSize* will be smaller than *MaximumByteCount* if the end of the stream is reached or to fit the data into the response buffer. If the stream is exactly the same size as *MaximumByteCount,* the client MUST issue another **RopReadStream** operation and get back zero bytes to determine that the end of the stream was reached.

If *ByteCount* a value other than 0xBABE, the value of *DataSize* is smaller than *ByteCount* if the end of the stream is reached or to fit the data into the response buffer. If the stream is exactly the same size as *ByteCount*, the client MUST issue another **RopReadStream** operation and get back zero bytes to determine that the end of the stream was reached.

In the case of a failure, *DataSize* SHOULD be zero, and the client SHOULD ignore any data it might get back from the server.

2.2.15.2.2 Data

Data contains the data read from the stream. *Data* MUST contain exactly *DataSize* BYTES.

2.2.16 RopWriteStream

RopWriteStream is used to write the stream of BYTES into a stream object. This ROP is valid only on **Stream** objects. For **Stream** objects opened on properties on **Folder** objects, information written to the stream requires a **RopCommitStream** before it is persisted. For all other **Stream** objects, the server will commit the stream when the client calls RopRelease. The syntax of this ROP is fully specified in [MS-OXCROPS].

2.2.16.1 Request Parameters Overview

2.2.16.1.1 DataSize

The value of *DataSize* is the number of BYTES in *Data.*

2.2.16.1.2 Data

Data contains the data to be written to the stream.

2.2.16.2 Response Parameters Overview

2.2.16.2.1 WrittenSize

The value of *WrittenSize* is the number of BYTES actually written.

2.2.17 RopCommitStream

RopCommitStream ensures that any changes made to a stream object are persisted in storage. This ROP is valid only on **Stream** objects. The syntax of this ROP is fully specified in [MS-OXCROPS].

2.2.17.1 Request Parameters Overview

None.

2.2.17.2 Response Parameters Overview

None.

2.2.18 RopGetStreamSize

RopGetStreamSize is used to retrieve the size of the stream.This ROP is valid only on **Stream** objects. The syntax of this ROP is fully specified in [MS-OXCROPS].

2.2.18.1 Request Parameter Overview

None.

2.2.18.2 Response Parameter Overview

2.2.18.2.1 StreamSize

The value of *StreamSize* is the number of BYTES in the stream. The maximum allowed stream size is 2^31 BYTES.

2.2.19 RopSetStreamSize

The **RopSetStreamSize** remote operation (ROP) sets the size of a stream.This ROP is valid only on **Stream** objects. The syntax of this ROP is fully specified in [MS-OXCROPS]. If the size of the stream is increased, then value of the extended stream MUST be zero. If the size of the stream is decreased, the information that extends past the end of the new size is lost.

2.2.19.1 Request Parameter Overview

2.2.19.1.1 StreamSize

The value of *StreamSize* is the size that the Stream will be set to. The maximum allowed stream size is 2^31 BYTES.

2.2.19.2 Response Parameter Overview

None.

2.2.20 RopSeekStream

RopSeekStream sets the seek pointer to a new location. This ROP is valid only on **Stream** objects. The new location is relative to either the beginning of the stream, the end of the stream, or the current seek pointer, which is specified by *Origin* parameter. **RopSeekStream**can also be used to get the current seek pointer, by setting *Origin* to 0x01 and *Offset* to zero. The syntax of this ROP is fully specified in [MS-OXCROPS].

2.2.20.1 Request Parameter Overview

2.2.20.1.1 Origin

The value of the *Origin* field MUST be one of the following values:

Value	Description
0x00	The new seek pointer is an offset relative to the beginning of the stream.
0x01	The new seek pointer is an offset relative to the current seek pointer location.
0x02	The new seek pointer is an offset relative to the end of the stream.

2.2.20.1.2 Offset

The value of the *Offset* field is a signed value representing the number of BYTES to offset the seek pointer from the origin. This value can be positive or negative.

2.2.20.2 Response Parameter Overview

2.2.20.2.1 NewPosition

The value of *NewPosition* is the number of BYTES from the beginning of the stream of the seek pointer.

2.2.21 RopCopyToStream

RopCopyToStream copies a specified number of bytes from the current seek pointer in the source stream to the current seek pointer in another destination stream.This ROP is valid only on **Stream** objects. The syntax of this ROP is fully specified in [MS-OXCROPS].

2.2.21.1 Request Parameter Overview

2.2.21.1.1 ByteCount

The value of *ByteCount* is the number of BYTES to copy from the source stream to the destination stream.

2.2.21.2 Response Parameter Overview

2.2.21.2.1 ReadByteCount

If *ReturnValue* is not **NullDestinationObject** [MS-OXCDATA] the value of *ReadByteCount* MUST be the number of BYTES read from the base object. Otherwise the value is undefined.

2.2.21.2.2 WrittenByteCount

If *ReturnValue* is not **NullDestinationObject** [MS-OXCDATA] the value of *WrittenByteCount* MUST be the number of BYTES written to the destination object. Otherwise the value is undefined.

2.2.22 RopProgress

RopProgress responses report the progress status of an asynchronous operation to the client. **RopProgress** requests are used to request the status of an asynchronous operations. They also MAY be used to abort an in-progress operation. If the abort flag sent by the client is non-zero**,** then the operation is aborted.

RopProgress responses MAY be used as a response to the following ROPs:

- RopCopyTo [MS-OXCPRPT]

- RopCopyProperties [MS-OXCPRPT]

- RopEmptyFolder [MS-OXCFOLD]

- RopDeleteMessages [MS-OXCFOLD]

- RopSetReadFlags [MS-OXCMSG]

- RopMoveCopyMessages [MS-OXCFOLD]

- RopMoveFolder [MS-OXCFOLD]

- RopCopyFolder [MS-OXCFOLD]

2.2.22.1 Request Parameter Overview

2.2.22.1.1 WantCancel

WantCancel is a **BYTE** sized field that MUST be zero if the client wants the current operation to continue. If the value if non-zero, the client is requesting that the server SHOULD attempt to cancel the operation.

2.2.22.2 Response Parameter Overview

2.2.22.2.1 CompletedTaskCount

CompletedTaskCount is a 4 **BYTE** field that SHOULD report an approximation of the number of tasks that the server has completed out of *TotalTaskCount* operations to complete the entire operation that the progress is being reported on.

2.2.22.2.2 TotalTaskCount

TotalTaskCount is a 4 **BYTE** field that MUST be greater than or equal to *CompletedTaskCount*. This field SHOULD represent an approximation of the number of tasks the server will complete for the operation that the progress is being reported on.

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 described in this document.

3.1.1.1 Property Object

A Property Object is an abstract concept that specifies the shared behaviors of **Messages**, **Folders**, **Logons** and **Attachments**.

3.1.1.2 Property Transactions

There are two different transaction models for property objects. Changes to properties on **Folders** and **Logons** are saved implicitly, whereas changes to **Message** objects are not saved until the client sends a **RopSaveMessage**. See [MS-OXCMSG] for more details on how **Messages** are saved.**Attachments** objects are not saved until the client sends a **RopSaveChangesAttach**. See [MS-OXCMSG] for more details on saving **Attachments**.

3.1.1.3 Named Properties

Named Properties are a set of mappings between **Property ID**s and **Property Names**. The client MAY cache the Property ID for any Property Names that it has queried using **RopGetPropertyIdsFromNames,** for the length of its session, but MUST requery the server before using these property IDs on a new session. A Property ID obtained for a Property Name is valid on any item within the **Logon**.

3.1.1.4 Streams

Changes made to properties that have been opened as a stream using **RopOpenStream** and written to using **RopWriteStream** MUST not be persisted to the **Property** object that property is on unless the client calls **RopCommitStream**.

3.1.2 **Timers**

None.

3.1.3 **Initialization**

There is no initialization specific to this protocol. Higher layers calling this protocol MUST obtain handles to the objects required by the message syntax specified in [MS-OXCROPS].

3.1.4 **Higher-Layer Triggered Events**

3.1.4.1 **When Client Needs to Query Data from an Object**

If any of the properties the client wants are Named Properties, the client MUST obtain the Property IDs for those properties. The Property IDs for those properties SHOULD be obtained using **RopGetPropertyIdsFromNames.** However, if the client has already obtained the Property IDs for the Property Names during this logon session on the same store as this object, the client MAY re-use the Property IDs returned from the server previously.

The client SHOULD then use **RopGetPropertiesSpecific** to query the value of those properties. The client SHOULD check the ReturnValue of the ROP, and the values of the properties returned for errors.

3.1.4.2 **When Client Needs to Set Data**

If any of the properties the client wants to set are Named Properties, the client MUST obtain the Property IDs for those properties. The Property IDs for those properties SHOULD be obtained using **RopGetPropertyIdsFromNames.** However, if the client has already obtained the Property IDs for the Property Names during this logon session on the same **Logon** as this object, the client MAY re-use the Property IDs returned from the server previously.

The client SHOULD then use **RopSetProperties** to set the value of those properties. The client SHOULD check the ReturnValue of the **ROP** and if *PropertyProblemCount* field in the response is non-zero, then the client SHOULD check the problem array, to verify which properties failed to be set.

3.1.4.3 **When the Cient Needs to Query Data Larger than Will Fit in a Single ROP**

The client MUST use a stream to read a property that came back with an error of **NotEnoughMemory** [MS-OXPROPS] in a **PropertyProblem [**MS-OXCDATA]. A client MAY use **RopOpenStream** to read a property even before attempting to read it using **RopGetPropertiesSpecific** if it expects the property to be big.

If the property the client wants to get is a Named Property, the client MUST obtain the Property ID for that property. The Property ID for the property SHOULD be obtained using **RopGetPropertyIdsFromNames**. However, if the client has already obtained the Property ID for the Property Names during this logon session on the same **Logon** as this object, the client MAY re-use the Property ID returned from the server previously.

The client MUST then use **RopOpenStream** to obtain a handle to a stream on the property. The client SHOULD use ReadOnly as the value for *OpenModeFlags* if the client is just going to read from the stream. The client MAY also use BestAccess or ReadWrite if the client later wishes to write to this stream. The client SHOULD check the ReturnValue of the ROP to verify that the handle was correctly retrieved.

The client MAY use **RopSeekStream** if the client does not want to read from the start of the stream. The client SHOULD then use one or more **RopReadStream** ROPs to read data from

the stream. The client SHOULD stop after it has read all the data it requires, or after **RopReadStream** returns zero as its **StreamDataSize.**

The client MUST use **RopRelease** after it is done with the stream object.

3.1.4.4 When the Client Needs to Set Data Larger than Will Fit in a Single RopSetProperties

If the property the client wants to set is a Named Property, the client MUST obtain the Property ID for that Property. The Property IDs for the property SHOULD be obtained using **RopGetPropertyIdsFromNames.** However, if the client has already obtained the Property ID for the Property Name during this logon session on the same **Logon** as this object, the client MAY re-use the Property ID returned from the server previously.

The client MUST then use **RopOpenStream** to obtain a handle to the stream on the property. The client MUST specify ReadWrite, Create, or BestAccess as the value for *OpenModeFlags*. The client SHOULD check the ReturnValue of the ROP to verify that the handle was correctly retrieved.

The client MAY use **RopSeekStream** if the client does not wish to write from the start of the stream. The client SHOULD then use one or more **RopWriteStream** ROPs to write data to the stream.

The client MUST use **RopCommitStream** to commit the changes to the property.

The client MUST use **RopRelease** once it is done with the stream object.

3.1.5 Message Processing Events and Sequencing Rules

With the exception of **RopProgress**, all the messages specified in [Section 2] of this protocol are all sent by the client and processed by the server. The client SHOULD process the **ROP Response Buffer** associated with each message it sends.

If the client specified *WantAsynchronous* for any of the following ROPs the server MAY respond with a **RopProgress** response.

- RopCopyTo [MS-OXCPRPT]

- RopCopyProperties [MS-OXCPRPT]

- RopEmptyFolder [MS-OXCFOLD]

- RopDeleteMessages [MS-OXCFOLD]

- RopSetReadFlags [MS-OXCMSG]

- RopMoveCopyMessages [MS-OXCFOLD]

- RopMoveFolder [MS-OXCFOLD]

- RopCopyFolder [MS-OXCFOLD]

The server MAY respond with a **RopProgress** response buffer to notify the client of its current progress. When the client receives a **RopProgress** response it MAY use the

CompletedTaskCount and *TotalTaskCount* to provide progress information to the user. The client MAYsend additional **RopProgress** requests to request additional status, or to abort the operation. In response to the clients **RopProgress** request, the server MUST respond with either the response to the original ROP, or another **RopProgress** response. If the client sends a ROP other than **RopProgress** to the server with the same logon before the asynchronous operation is complete the server MUST abort the asynchronous operation and respond to the new ROP.

3.1.6 **Timer Events**

None.

3.1.7 **Other Local Events**

None.

3.2 **Server Details**

3.2.1 **Abstract Data Model**

The Server Abstract Data Model is the same as the Client Abstract Data Model. See section 3.1.1.

3.2.2 **Timers**

None.

3.2.3 **Initialization**

None.

3.2.4 **Higher-Layer Triggered Events**

None.

3.2.5 **Message Processing Events and Sequencing Rules**

3.2.5.1 Processing RopGetPropertiesSpecific

Provides access to property values on a **MessagingObject** on the server. The server MUST return the values for all properties on the object, including those set by any client, server, or computed properties. If the *WantUnicode* flag is used, the server MUST return string properties which are requested without a specified type (**PtypUnspecified**) in Unicode format.

If the *WantUnicode* flag is not used, the server MUST return string properties which are requested without a specified type (**PtypUnspecified**) in **MBCS** format.

Properties requested with a specific string type MUST be returned using that type.

For properties on **Message** objects the code page used for strings in **MBCS** format MUST be the code page set on the message when it was opened if any, otherwise the code page of the **Logon**.

For properties on **Attachment** objects the code page used for strings in **MBCS** format MUST be the code page set on the parent message when it was opened if any otherwise the code page of the **Logon**.

All other objects the code page used for strings in **MBCS** format MUST be the code page of the **Logon**.

In the case of a binary, object or string property the server MAY return **NotEnoughMemory** [MS-OXPROPS] for that property to indicate that the value is to large to be returned using this Rop. The client MAY use **RopOpenStream** to retrieve the values of such large properties.

This ROP SHOULD be supported by only **Message**, **Folder**, **Attachment** and **Logon** objects.

3.2.5.2 Processing RopGetPropertiesAll

Provides access to all property values on a **Messaging Object** on the server.This ROP works the same as **RopGetPropertiesSpecific** except that it MUST return the values for all properties on the object.

3.2.5.3 Processing RopGetPropertiesList

The server MUST return the list of all properties currently set on the object.

This ROP SHOULD be supported for only **Message**, **Folder**, **Attachment** and **Logon** objects.

3.2.5.4 Processing RopSetProperties

The server MUST modify the value of the properties of the object.

For **Message** objects, the new value of the properties MUST be made available immediately when using the same handle. If the client uses the same handle to read those same properties using **RopGetPropertiesSpecific** or **RopGetPropertiesAll**, then the modified value MUST be returned. However, the modified value MUST NOT be persisted to the database until a successful **RopSaveChangesMessage** is issued.

For **Attachment** objects, the new value of the properties MUST be made available immediately when using the same handle. However, the modified value MUST NOT be persisted to the database until a successful **RopSaveChangesAttachment** is issued.

For **Folder** and **Logon** objects, the new value of the properties MUST be persisted immediately without requiring another rop to commit it.

This ROP SHOULD be supported for only **Messages**, **Folders**, **Attachments** and **Logon** objects.

3.2.5.5 Processing RopDeleteProperties

Removes the value of a set property. If the server returns success it MUST NOT have a valid value to return to a client that asks for the value of this property.

The server MUST remove the value of the property from the object.

For **Message** objects, the new value of the properties MUST be made available immediately when using the same handle. In other words, if the client uses the same handle to read those same properties using **RopGetPropertiesSpecific** or **RopGetPropertiesAll**, then the modified value MUST be returned. However, the modified value MUST NOT be persisted to the database until a successful **RopSaveChangesMessage** is issued.

For **Attachment** objects, the new value of the properties MUST be made available immediately when using the same handle. However, the modified value MUST NOT be persisted to the database until a successful **RopSaveChangesAttachment** is issued.

For **Folder** and **Logon** objects, the new value of the properties MUST be persisted immediately without requiring another rop to commit it.

This ROP SHOULD be supported for only **Messages**, **Folders**, **Attachments** and **Logon** objects.

3.2.5.6 Processing RopQueryNamedProperties

The server MUST return the list of all NamedProperties and their PropertyIds.

If a property guid is specified, the server MUST only return the named properties with that guid.

If a property guid is not specified, the server MUST return all named properties.

This ROP SHOULD only be supported for **Messages**, **Folders**, and **Attachments**.

3.2.5.7 Processing RopCopyProperties

The server MUST copy the properties specified from the source object to the destination object.

If the move flag is specified, the server MUST delete the properties from the original object.

In the case of **Attachments** and **Messages**, the changes on either source or destination MUST not be persisted until **RopSaveChangesMesssage** or **RopSaveChangesAttachment** is successfully issued.

In the case of **Folders**, the changes on the source and destination MUST be immediatelly persisted.

If the client requests asynchronous execution, then the server MAY execute this ROP asynchronously. See [Section 3.1.5].

This ROP SHOULD be supported for only **Message**, **Attachment**, and **Folder** objects.

3.2.5.8 Processing RopCopyTo

This ROP works in the same way as **RopCopyProperties** except that it MUST copy all writtable properties on the source to the destination except for the properties specified.

The following error codes MAY be returned:

Name	Value	Meaning
NotSupported	0x80040102	The source object and destination object are not compatible with each other for the copy operation. The source object and destination object need to be of the same type, and MUST be a **Message**, **Folder**, or **Attachment**.
MessageCycle	0x00000504	The source object performing the copy or move operation directly or indirectly contains the destination object.
CollidingNames	0x80040604	A sub-object cannot be copied because there is already a sub-object existing in the destination object with the same display name (PidTagDisplayName) as the sub-object to be copied.

3.2.5.9 Processing RopGetNamesFromPropertiesIds

This ROP SHOULD apply to only **Folder**, **Messages**, **Attachments** and **Logon** objects.

For each property ID in the *PropertyIds* parameter, the server MUST perform the following:

1. If the property ID value is less than 0x8000, the associated **PropertyName** [MS-OXCDATA] MUST be composed:

    a. A **GUID** [MS-OXCDATA] of *PS-MAPI* [MS-OXPROPS] .

    b. A **Kind** [MS-OXCDATA] of 0x00.

    c. An **LID** [MS-OXCDATA] value equal to the property ID entry.

2. For property ID values that have an associated **PropertyName** [MS-OXCDATA], the server MUST return the **PropertyName** associated with the PropertyId.

3. For property ID values that do not have an associated **PropertyName**, the associated name MUST be composed:

    a. A **Kind** [MS-OXCDATA] of 0xFF. There is no other return data for this entry. The whole operation MUST return with the partial success code **ErrorsReturned** [MS-OXCDATA].

3.2.5.10 Processing RopGetPropertyIdsFromNames

The server MUST validate all name structures passed by the client as specified in section 2.2.2.14.

If the *Count* parameter is zero, and the **RopGetPropertyIdsFromNames** is acting on a **Logon** object, the server MUST enumerate all PropertyNames associated with Property IDs. Otherwise, the server MUST, for each entry in the *NamedPropertyArray* parameter:

1. If the **GUID** [MS-OXCDATA] is equal to *PS-MAPI*, the associated return value is obtained from the *LID*.

2. Otherwise, if the **GUID** [MS-OXCDATA] is equal to *PS-INTERNET-HEADERS* [MS-OXPROPS] and **Kind** [MS-OXCDATA] is 0x01, coerce the *Name* value to all lower case.

3. Find the Property ID associated with the PropertyName which precisely matches the **Kind** [MS-OXCDATA] and ID values from the name entry. The ID value is the value of *Name* if *Kind* is equal to 0x01 or is the value of **LID** [MS-OXCDATA] if **Kind** [MS-OXCDATA] is equal to 0x00.

4. For unfound rows, the associated return value MUST be 0xFFFF unless the *Flags* parameter has the *Create* flag bit set and the user has permission to create new entries and any enforceable quota hasn't yet been reached and there are fewer than 32,767 PropertyNames already associated with Property IDs, a new PropertyID is associated with this **PropertyName**. The newly assigned ID MUST be unique in that it MUST NOT be assigned to another other **PropertyName**, and MUST NOT be equal to 0xFFFF. The newly assigned ID MUST be greater than 0x8000.

The following error code can be returned.

Name	Value	Meaning
ErrorsReturned	0x80040380	Some names could not be mapped to an Id.

3.2.5.11 Processing RopOpenStream

When the client sends the server a **RopOpenStream** request, the server MUST parse the request as specified in [MS-OXCROPS] and section 2.2.12.1. The server MUST respond with a **RopOpenStream** response as specified in section 2.2.12.2.

RopOpenStream provides access to a property in the form of a **Stream**. The object returned by this ROP can then be used on subsequent ROPs, such as **RopReadStream**, **RopWriteStream** and **RopCommitStream**. See specific ROPs for information of which operate on streams.

This rop SHOULD only be supported for **Message**, **Attachment** and **Folder** objects.

The implementation of **RopOpenStream** MAY allocate some temporary resource on the server to represent the link between the **Folder** object and the **Folder** on the database. The client MUST call **RopRelease** on the **Folder** object to free this resource.

The server MUST store the location of the seek pointer until the client calls **RopRelease**. The seek pointer MUST be initialized as specified in the request. The stream MUST be prepopulated with the current value of the property specified in the request. When the client calls **RopRelease** on the **Stream**, and the property the **Stream** is acting on is not a **Folder**, the server MUST commit the **Stream**, copying the **Stream** back into the property.

The following error codes can be returned.

Name	Value	Meaning
NotFound	0x8004010F	The property tag does not exist for the object and it cannot be created because **Create** was not specified in *OpenModeFlags*.

3.2.5.12 Processing RopReadStream

The server MUST read less or equal to the amount of data requested.

The server MAY read less than the amount requested.

This ROP SHOULD only be supported for **Stream** objects.

The server MUST reply with data read from the stream reading forward from the current seek pointer and place it in the response. The seek pointer MUST be moved forward the same number of BYTES as was read from the **Stream**.

3.2.5.13 Processing RopWriteStream

The server MAY write less than the amount requested, in which case **StreamSizeError** SHOULDbe returned.

This ROP SHOULD only be supported for **Stream** objects.

The server MUST store the data specified in the request into the stream buffer, writing forward starting at the current seek pointer.The seek pointer MUST be moved forward the same number of BYTES as was written to the **Stream.** Writing to this stream MUST not change the value of the property specified in the **RopOpenStream** request.

The following error codes can be returned:

Name	Value	Description
StreamSizeError	0x80030070	The write will exceed the maximum stream size.
STG_E_ACCESSDENIED	0x80030005	Write access is denied.

3.2.5.14 Processing RopCommitStream

This rop SHOULD only be supported for **Stream** objects.

The server MUST set the property specified in the **RopOpenStream** request that opened this stream with the data from the stream.

The following error code can be returned

Name	Value	Description
STG_E_ACCESSDENIED	0x80030005	Write access is denied.

3.2.5.15 Processing RopGetStreamSize

RopGetStreamSize MUST return the current size of the **Stream**.

This ROP SHOULD only be supported for **Stream** objects.

3.2.5.16 Processing RopSetStreamSize

This ROP MUST set the current size of the **Stream**.

This ROP SHOULD only be supported for **Stream** objects.

3.2.5.17 Processing RopSeekStream

This ROP SHOULD only be supported for **Stream** objects.

The value of the seek point associated with this **Stream** MUST be modified according to the request. If the client requests the seek pointer be moved beyond 2^31 BYTES, the server MUST return **StreamSeekError**. If the client requests the seek pointer be moved beyond the end of the stream, the stream is extended, and zero filled to the new seek location.

The following error codes can be returned:

Name	Value	Description
StreamSeekError	0x80030019	Tried to seek to offset before the start, or beyond the max stream size of 2^31.
StreamInvalidParam	0x80030057	The value of *Origin* is invalid.

3.2.5.18 Processing RopCopyToStream

This rop SHOULD only be supported for **Streams**.

The position of both the source and destination streams MUST be moved forward the same number of BYTES as was copied.

The server MUST read the number of BYTES requested from the source **Stream**, and write those bytes into the destination **Stream**.

The following error code can be returned.

Name	Value	Description
DestinationNullObject	0x00000503	**Destination** object does not exist. Only if the *ReturnValue* is **DestinationNullObject**, which means that the destination object passed does not exist anymore, then the server sends the *DestinationHandleIndex* back in 4 bytes.

3.2.5.19 Processing RopProgress

If the server is not done with the asynchronous operation it SHOULD respond with a **RopProgress** response as specified in section 2.2.21, or the server MAY wait until the operation is complete and respond to the next **RopProgress** with a ROP response corresponding to the original request. If the server receives a **RopProgress** after it has completed the operation, it MUST respond with a ROP response corresponding to the original request.If *WantCancel* is non-zero the server SHOULD abort the current asynchronous operation, the server MAY choose to finish the operation.

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. Please note that the examples listed here only show the relevant portions of the specified ROPs; this is not the final byte sequence which gets transmitted over the wire. Also note that the data for a multi-byte field appear in **little-endian** format, with the bytes in the field presented from least significant to most significant. Generally speaking, these ROP requests are packed with other ROP requests, compressed and packed in one or more RPC calls according to the specification in [MS-OXCROPS]. These examples assume the client has already successfully logged onto the server and opened the folder they wish to modify the rules on. For more details, see [MS-OXCROPS].

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

0080: 45 4d 53 <u>4d</u> 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. Here, the underlines byte "<u>4d</u>" (01001101) is located 0x83 bytes (131 bytes) from the beginning of the buffer. The dash between eighth byte ("44") and ninth byte ("4c") bytes has no semantic value, and serves only to distinguish the eight byte boundary for readability purposes.

Such a byte sequence is then followed by one or more lines interpreting 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.

The following example shows how a "**Property Tag**" and its "Property Value" are represented in a buffer and interpreted directly from it (according to the Property Buffer format specified in [MS-OXCDATA]). The property tag appears in the buffer in little-endian format.

0021: 03 00 76 66 0a 00 00-00

       PropertyTag: 0x66760003 (**PidTagRuleSequence** [MS-OXCPROPS])

       PropertyValue: 10

Generally speaking interpreted values will be shown in their native format, interpreted appropriately from the raw byte sequence as described in the appropriate section. Here, the byte sequence "0a 00 00 00" has been interpreted as a **ULONG** with a value of 10 because the type of the **PidTagRuleSequence** property is **ULONG**.

4.1 Getting PropertyIds

The following example describes the contents of the ROP request and response buffers for a successful *RopGetPropertyIdsFromNames* operation as described insection 2.2.12

4.1.1 Client Request Buffer

In this example the client calls server to get Property Ids for 2 named properties it wants to create on a message.

A complete ROP request buffer is formatted as follows:

```
0000: 56 00 00 02 02 00 01 02-20 06 00 00 00 00 00 c0
0010: 00 00 00 00 00 00 46 14-54 00 65 00 73 00 74 00
0020: 50 00 72 00 6f 00 70 00-31 00 00 00 01 02 20 06
0030: 00 00 00 00 00 c0 00 00-00 00 00 00 46 14 54 00
0040: 65 00 73 00 74 00 50 00-72 00 6f 00 70 00 32 00
0050: 00 00
```

The first three bytes of the buffer refer to the *RopId*, *LogonIndex*, and *HandleIndex* fields of
RopGetPropertyIdsFromNames as defined in [MS-OXCROPS].

0000: 56 00 00

> RopId: 0x56 (RopGetPropertyIdsFromNames)
>
> LogonIndex: 0x00
>
> HandleIndex: 0x00

The next three bytes refer to the *InputFlag* and *NameIdDataCount* fields defined in section
2.2.12.1. For more details on property buffer format, see [MS-OXCDATA].

0003: 02 02 00

> InputFlag: 0x02.   CreateIfExits flag, where Property Name that does not already
> have an associated Property ID the server will attempt to create
> one.
>
> DataCount: 0x0002.   Two properties in NameIdDataArray follow.

The remaining bytes form the *NameIdDataArray.* Each row in NameIdDataArray contains
Kind (I byte), *GUID (16 bytes)*, *NameCount* (if Type is *MNID_STRING*) and *NameID (long
or variable length string)* of the property. For details, see [MS-OXCDATA].

0006: 01 02-20 06 00 00 00 00 00 c0

0010: 00 00 00 00 00 00 46 14-54 00 65 00 73 00 74 00

0020: 50 00 72 00 6f 00 70 00-31 00 00 00 01 02 20 06

0030: 00 00 00 00 00 c0 00 00-00 00 00 00 46 14 54 00

0040: 65 00 73 00 74 00 50 00-72 00 6f 00 70 00 32 00

0050: 00 00

Property Name 1:

> Kind: 0x01 MNID_STRING
>
> PropertyGUID: 00062002-0000-0000-c000-000000000046
>
> NameCount: 0x14
>
> NameString: TestProp1

Property Name 2:

> Kind: 0x01 MNID_STRING
>
> PropertyGUID: 00062002-0000-0000-c000-000000000046
>
> NameCount: 0x14
>
> NameString: TestProp2

4.1.2 **Server Response to Client Request**

`0000:`56 00 00 00 00 00 02 00-3e 86 3f 86

The first six bytes of the response buffer contain the *RopId*, *HandleIndex*, and *ReturnValue* described in [MS-OXCROPS].

0000:56 00 00 00 00 00

      RopId: 0x56 (RopGetPropertyIdsFromNames)

      HandleIndex: 0x00

      ReturnValue: 0x00000000 (**Success** [MS-OXCDATA])

The next two bytes is for the *PropertyTagCount* of property IDs that are packed in the buffer as described in section 2.2.12.2.1.

      PropertyTagCount: 0x0002 (2 properties in the property tag array)

The remaining is *PropertyTagArray* where each entry in the array is four byte property ID as described in section 2.2.12.2.2.

0009:3e 86 3f 86

      Property ID: 0x863e

      Property ID: 0x863f

4.2 **Setting Properties**

The following example describes the contents of the ROP request and response buffers for a successful *RopSetProperties* operation as described in section 2.2.5.

4.2.1 **Client Request Buffer**

In this example the client is setting values for 4 properties on a message object, 2 named properties and 2 standard properties. The PropertyIds for the named properties were gotten from server by calling *RopGetPropertyIdsFromNames*, see section 4.1 for details.

A complete ROP request buffer is a variable length sequence, with seven required bytes and followed by property value array. An example of the request buffer follows:

`0000:` 0a 00 00 33 00 04 00 0b-00 3e 86 00 03 00 3f 86

`0010:` 62 00 00 00 00 1f00 3d-00 00 00 1f 00 1d 0e 48

`0020:`00 65 00 6c 00 6c00 6f-00 2000 57 00 6f 00 72

0030: 00 6c 00 64 00 00 00

The first three bytes of the buffer refer to the *RopId*, *LogonIndex*, and *HandleIndex* fields of *RopSetProperties* as defined in [MS-OXCROPS].

0000: 0a 00 00

    RopId: 0x0a (RopSetProperties)

    LogonIndex: 0x00

    HandleIndex: 0x00

The next four bytes refer to the *PropertyValueSize* and *PropertyValueCount* fields of RopSetProperties defined in section 2.2.5.1. For more details on property buffer format, see [MS-OXCDATA].

0003: 33 00 02 00

    PropertySize: 0x0033. Size of Count and Array that follows.

    PropertyCount: 0x0004. Four properties in property array.

The remaining bytes constitute *PropertyValues*, where each row in array consists of Property Tag and Value. A property tag is a 32-bit number that contains a property type in bits 0 through 15 and a unique property identifier in bits 16 through 31 as shown in the following illustration.

                                  0008: 0b-00 3e 86 00 03 00 3f 86

0010: 62 00 00 00 00 1f00 3d-00 00 00 1f 00 1d 0e 48

0020: 00 65 00 6c 00 6c00 6f-00 2000 57 00 6f 00 72

0030: 00 6c 00 64 00 00 00

Property 1 (**0008:** 0b-00 3e 86 00)

PropertyTag: 0x863E000B (PropertyID: 0x863E->TestProp1,PropertyType: 0x000B-> PtypBoolean)

PropertyValue: 00 (False)

Property 2 (**000D:** 03 00 3f 8662 00 00 00)

PropertyTag: 0x863F0003 (PropertyID: 0x863F->TestProp2,PropertyType: 0x0003-> PtypInteger32)

PropertyValue: 0x00000062 (98 in decimal)

Property 3 (**0014:**00 1f00 3d-00 00)

PropertyTag: 0x003D001F (PropertyID: 0x003D->PidTagSubjectPrefix, PropertyType: 0x001F->PtypString)

PropertyValue: 0x0000 (Empty String)

Property 4 (**001A:**00 1f 00 1d 0e 48 00 65 00 6c 00 6c00 6f-00 2000 57 00 6f 00 72 00 6c 00 64 00 00 00)

PropertyTag: 0x001D001F (PropertyID: 0x001D->PidTagNormalizedSubject, PropertyType: 0x001F-> PtypString)

PropertyValue: 0e 48 00 65 00 6c 00 6c00 6f-00 2000 57 00 6f 00 72 00 6c 00 64 (Hello World)

4.2.2 Server Response to Client Request

0000:0a 00 00 00 00 00 00 00

The first six bytes of the response buffer contain the *RopId*, *HandleIndex*, and *ReturnValue* fields.

> RopId: 0x0a (RopSetProperties)
>
> HandleIndex: 0x00
>
> ReturnValue: 0x00000000. (ecNone: Success)

The final two bytes in the response buffer is the *PropertyProblemCount* field described in section 2.2.5.2.1.

0006: 00 00

> PropertyProblemCount: 0x0000. Count of Problem property that follows. 0 indicates that all properties were set successfully.

4.3 Getting Properties

The following example describes the contents of the ROP request and response buffers for a successful *RopGetPropertiesSpecific* operation as described in section 2.2.2.

4.3.1 Client Request Buffer

In this example client is requesting property values for specific properties from the server. The first two are named properties and third one is standard property. The PropertyIds for the

named properties were gotten from server by calling *RopGetPropertyIdsFromNames*, see section 4.1 for details.

A complete ROP request buffer is a variable length sequence, with nine required bytes and followed by property tag array. An example of the request buffer follows:

```
0000:07 00 00 00 00 01 00 03-00 0b 00 3e 86 03 00 3f
0010: 86 02 01 e2 65
```

The first three bytes refer to the *RopId*, *LogonIndex*, and *HandleIndex* fields described in [MS-OXCROPS].

0000:07 00 00

> RopId: 0x07 (RopGetPropertiesSpecific)
>
> LogonIndex: 0x00
>
> HandleIndex: 0x00

The next six bytes of the request buffer are the *PropertySizeLimit, WantUnicode*, and *PropertyTagCount*fields described in section 2.2.2.1.

0003:00 00 01 00 03-00

> PropertySizeLimit: 0x0000. No prop size set; maximum limit is default
>
> WantUnicode: 0x0001. Non-zero means Unicode
>
> PropertyValueCount: 0x0003. 3 properties tags in array

The remaining bytes constitute *PropertyTags* as described in section 2.2.2.1.1.4.

0009: 0b 00 3e 86 03 00 3f 86 02 01 e2 65

PropertyTag1: 0x863E000B(PropertyID: 0x863E->TestProp1,PropertyType: 0x000B->PtypBoolean)

PropertyTag2: 0x863F0003 (PropertyID: 0x863F->TestProp2,PropertyType: 0x0003->PtypInteger32)

PropertyTag3: 0x65E20102 (PropertyID: 0x65E2->PidTagChangeKey,PropertyType: 0x0102->PtypBinary)

4.3.2 Server Response to Client Request

```
0000:07 00 00 00 00 00 01 00-00 00 62 00 00 00 0a 0f
0010:01 04 80
```

The first six bytes of the response buffer contain the *RopId*, *HandleIndex*, and *ReturnValue* fields.

0000:07 00 00 00 00 00

> RopId: 0x07 (RopGetPropertiesSpecific)
>
> HandleIndex: 0x00
>
> ReturnValue: 0x00000000 (**Success** [MS-OXCDATA])

The remaining bytes in the response buffer are for the *PropertyValues*. The first byte of *PropertyValues* is the **Flag** value. Then three **FlaggedPropertyValue**s [MS-OXCDATA] each consisting of a **Flag** and a **PropertyValue** [MS-OXCDATA] The order of properties MUST be same as in the request buffer.

0006:0000 01 00 62 00 00 00 0a 0f01 04 80

```
Flag: 0x01 (Non-zero indicates there was an error in atleast
one of the property values)
```

Property 1:

Flag: 0x00

> PropertyValue: 0x00 (False)

Property 2:

Flag: 0x00

> PropertyValue: 0x00000062 (98 in decimal)

Property 3:

> Flag: 0x0a (Indicates that the PropertyValue field will be an error code)
>
> PropertyValue: 0x8004010f (**NotFound** [MS-OXCDATA])

4.4   Working with Streams

The following sections give examples of how to set value for a property using streams. The ROPs covered are **RopOpenStream**, **RopSetStreamSize**,**RopWriteStream** and **RopCommitStream**.

4.4.1   Opening a Stream

The following example describes the contents of the ROP request and response buffers for a successful **RopOpenStream** as described in2.2.14.

4.4.1.1 Client Request Buffer

A complete ROP request buffer is nine bytes in length. An example of the request buffer follows:

```
0000:2b 01 00 01 02 01 9a 0e-01
```

The first four bytes of the buffer refer to the *RopId*, *LogonIndex*, *HandleIndex* and *StreamHandleIndex* fields of **RopOpenStream** as defined in [MS-OXCROPS].

```
0000:2b 01 00 01
```

> RopId: 0x2b (RopOpenStream)
>
> LogonIndex: 0x01
>
> HandleIndex: 0x00. Index in the *HandleArray* for the base object used by this ROP
>
> StreamHandleIndex: 0x01. Index in the server object handle table for the object
created by this ROP.

The next five bytes refer to the *PropertyTag* and *OpenModeFlag* defined in section 2.2.14.1.2.

```
0004:02 01 9a 0e-01
```

> PropertyTag: 0e9a0102 (PidTagExtendedRuleMessageCondition [MS-OXCPROPS])
>
> OpenModeFlag: 0x0001. ReadWrite Mode

4.4.1.2 Server Response to Client Request

```
0000:2b 01 00 00 00 00 15 2e-00 00
```

The first six bytes of the response buffer contain the *RopId*, *HandleIndex*, and *ReturnValue* fields.

```
0000:2b 01 00 00 00 00
```

> RopId: 0x2b (RopOpenStream)
>
> HandleIndex: 0x01
>
> ReturnValue: 0x00000000 (**Success** [MS-OXCDATA])

The last four bytes in the response buffer is the *StreamSize* field as described in section 2.2.14.2.

```
0006:15 2e-00 00
```

> StreamSize: 0x00002e15 (11797)

4.4.2 **Writing to the stream**

The following example describes the contents of the ROP request and response buffers for a successful **RopWriteStream** operation as described in section 2.2.16.

4.4.2.1 Client Request Buffer

A complete ROP request buffer is a variable length buffer, formatted as follows:

```
0000: 2d 01 01 15 2e 00 00 61-6e 20 61 6c 77 61 79 73
0010: 20 72 65 73 74 6f 72 65-20 74 68 65 20 6c 6f 6f
0020: 6b 20 6f 66 20 79 6f 75-72 20 64 6f 63 75 6d 65
```

The first three bytes of the buffer refer to the *RopId*, *LogonIndex*, and *HandleIndex* fields of RopWriteStream as defined in [MS-OXCROPS].

```
0000: 2d 01 01
```

RopId: 0x2d (RopWriteStream)

LogonIndex: 0x01

HandleIndex: 0x01

The next two bytes in the response buffer is the *DataSize* field as described in section 2.2.16.1.1.

```
0006: 15 2e
```

DataSize: 0x2e15 (11797)

The remaining bytes constitute the stream data. The buffer shown above is truncated and all of the stream data is not shown.

```
Data: 00 00 61-6e 20 61 6c 77 61 79 73 20 72 65 73 74 6f 72
      65-20 74 68 65 20 6c 6f 6f 6b 20 6f 66 20 79 6f 75-72 20
      64 6f 63 75 6d 65..........
```

4.4.2.2 Server Response to Client Request

```
0000: 2d 01 00 00 00 00 15 2e
```

The first six bytes of the response buffer contain the *RopId*, *HandleIndex*, and *ReturnValue* fields.

```
0000: 2d 01 00 00 00 00
```

RopId: 0x2d (RopWriteStream)

HandleIndex: 0x01

ReturnValue: 0x00000000 (**Success** [MS-OXCDATA])

The last two bytes contain the *WrittenSize* field described in section 2.2.16.2.1.

0006:15 2e

WrittenSize: 0x2e15 (11797)

4.4.3 **Committing a Stream**

The following example describes the contents of the ROP request and response buffers for a successful **RopCommitStream** operation as described in section 2.2.17.

4.4.3.1 Client Request Buffer

There MUST be a stream open before this ROP is called. A complete ROP request buffer for **RopCommitStream** is a three byte sequence formatted as follows:

0000:5d 01 01

The three bytes refer to the *RopId*, *LogonIndex*, and *HandleIndex* fields described in [MS-OXCROPS].

RopId: 0x5d (RopCommitStream)

LogonIndex: 0x01

HandleIndex: 0x01

4.4.3.2 Server Response to Client Request

0000:5d 01 00 00 00 00

The six bytes of the response buffer contain the *RopId*, *HandleIndex*, and *ReturnValue* fields.

RopId: 0x5d (RopCommitStream)

HandleIndex: 0x00

ReturnValue: 0x00000000 (**Success** [MS-OXCDATA])

4.5 **Asynchronous Progress**

The following example describes the contents of the ROP request and response buffers for a successful **RopProgress** as described in section 2.2.20. In this example user is trying to empty a **Folder**, which has 729 items in it and **RopEmptyFolder** will represent the async operation that **RopProgress** reports progress for.

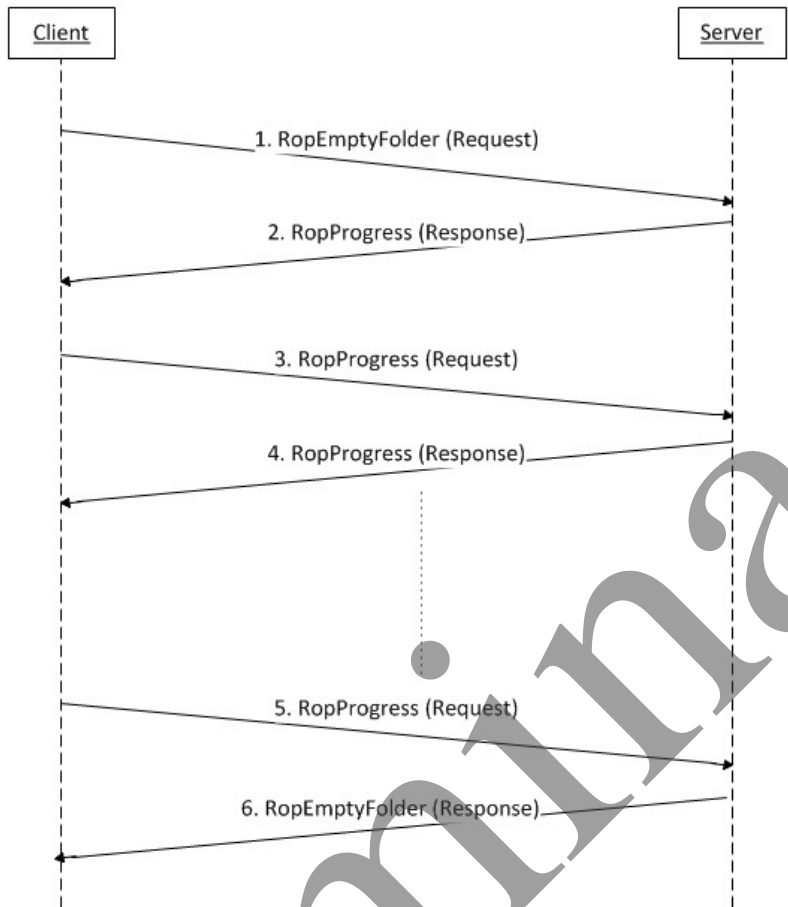The sequence in which ROPs get passed between client and server is shown in Figure 1.

Figure 1: Sequence in which ROP are passed between client and server

1. Client sends a **RopEmptyFolder** request to the server as described in [MS-OXCROPS].
   The request buffer is formatted as follows:

   0000: 58 00 00 01 00

   RopId: 0x58 (RopEmptyFolder)

   LogonIndex: 0x00

   HandleIndex: 0x00

   WantAsynchronous: 0x01 (TRUE)

   WantDeleteEverything: 0x00 (FALSE)

2. Server responds to request by sending **RopProgress** response buffer. The response
   buffer is formatted as follows:

   0000: 50 00 00 00 00 00 00 1d-00 00 00 d9 02 00 00

   RopId: 0x50 (RopProgress)

HandleIndex: 0x00

ReturnValue: 0x00000000 (**Success** [MS-OXCDATA])

LogonIndex: 0x00

CompletedTaskCount: 0x0000001d (29 in decimal)

TotalTaskCount: 0x000002d9 (729 in decimal)

3. Now client sends a **RopProgress** request buffer asking server for the how much progress has been made. The response buffer is formatted as follows:

```
0000:  50 00 00 00
```

RopId: 0x50 (RopProgress)

LogonIndex: 0x00

HandleIndex: 0x00

WantCancel: 0x00 (FALSE)

4. Server responds to request by sending **RopProgress** response buffer. The response buffer is formatted as follows:

```
0000: 50 00 00 00 00 00 00 3b 00 00 00 d9 02 00 00
```

RopId: 0x50 (RopProgress)

HandleIndex: 0x00

ReturnValue: 0x00000000 (ecNone: Success)

LogonIndex: 0x00

CompletedTaskCount: 0x0000003b (59 in decimal)

TotalTaskCount: 0x000002d9 (729 in decimal)

5. Client keeps sending **RopProgress** requests and server keeps sending **RopProgress** response buffer telling client the current progress status.

Finally when server has completed the **RopEmptyFolder** operation, then instead of sending a RopProgress resposnse buffer, it sends **RopEmptyFolder** response buffer back. The following is last RopProgress request that client makes:

```
0000:  50 00 00 00
```

RopId: 0x50 (RopProgress)

LogonIndex: 0x00

HandleIndex: 0x00

WantCancel: 0x00 (FALSE)

6. Server responds by sending **RopEmptyFolder** response buffer back formatted as follows:

`0000:` 58 00 00 00 00 00 00

RopId: 0x58 (RopEmptyFolder)

HandleIndex: 0x00

ReturnValue: 0x00000000 (**Success** [MS-OXCDATA])

5 Security

5.1 Security Considerations for Implementers

There are no special security considerations specific to the Property and Stream Object protocol. General security considerations pertaining to the underlying RPC-based transport apply (see [MS-OXCROPS]).

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:

- Office 2003 with Service Pack 3 applied
- Exchange 2003 with Service Pack 2 applied
- Office 2007 with Service Pack 1 applied
- Exchange 2007 with Service Pack 1 applied

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.

7   Index