[MS-OXCDATA]: Data Structures Protocol Specification

Intellectual Property Rights Notice for Protocol Documentation

- **Copyrights**. This protocol 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 protocols, and may distribute portions of it in your implementations of the protocols or your documentation as necessary to properly document the implementation. This permission also applies to any documents that are referenced in the protocol documentation.

- **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 protocols. Neither this notice nor Microsoft's delivery of the documentation grants any licenses under those or any other Microsoft patents. However, the protocols may be covered by Microsoft's Open Specification Promise (available here: http://www.microsoft.com/interop/osp/default.mspx). If you would prefer a written license, or if the protocols are not covered by the OSP, patent licenses are available by contacting protocol@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.

Preliminary Documentation. This documentation is preliminary documentation for these protocols. Since the documentation may change between this preliminary version and the final version, there are risks in relying on preliminary documentation. To the extent that you incur additional development obligations or any other costs as a result of relying on this preliminary documentation, you do so at your own risk.

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.
Microsoft Corporation	April 25, 2008	0.2	Revised and updated property names and other technical content.

Table of Contents

1 Introduction

Certain data structures appear repeatedly in different **remote operations (ROPs)** and property values, and in both store and address book protocols.

The Data Structures Protocol specifies certain common data structures that are used repeatedly in the ROPs specified in the Remote Operations (ROP) List and Encoding Protocol and in the Office Exchange Protocols Master Property List. This protocol includes structure layouts and semantics.

1.1 Glossary

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

entry ID
LongTermID
Personal Information Manager (PIM)
remote operation (ROP)
X500 DN

The following terms are specific to this document:

double-byte character set (DBCS): A charset, such as iso-2022-jp, in which characters are encoded as either 1 or 2 bytes.

multiple-byte character set (MBCS): A charset, such as iso-2022-jp, in which more than 1 byte is required to encode at least some characters.

single-byte character set (SBCS): A charset, such as US-ASCII, in which all characters are encoded as a single byte.

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-DTYP] Microsoft Corporation, "Windows Data Types", March 2007, http://go.microsoft.com/fwlink/?LinkId=111558.

[MS-NSPI] Microsoft Corporation, "Name Service Provider Interface (NSPI) Protocol Specification", April 2008.

[MS-OAUT] Microsoft Corporation, "OLE Automation Protocol Specification", March 2007, http://go.microsoft.com/fwlink/?LinkId=112419.

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

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

[MS-OXOAB] Microsoft Corporation, "Offline Address Book (OAB) Format and Schema Protocol Specification", April 2008.

[MS-OXOABK] Microsoft Corporation, "Address Book Object Protocol Specification", April 2008.

[MS-OXOCNTC] Microsoft Corporation, "Contact Object Protocol Specification", April 2008.

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

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

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

[MS-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 Structure Overview (Synopsis)

The Data Structures Protocol specifies several commonly used data structures. These structures are primarily concerned with property values, folder and message object identifiers, and folder queries.

There are some apparent redundancies; for example, **EntryIds** are specified in a couple of different ways in section 2.2. This is because information is formatted differently in different

contexts. For example, store EntryIds are formatted differently in the context of a **remote operation (ROP)** than in the context of a binary property value created by clients.

As a rule, integers in the data structures here specified are transmitted in little-endian byte order, with the *least significant byte* first. But when individual bits within a byte field are specified, they are numbered *starting with the most significant bit*. Therefore, in a 1-byte field, bit 0 is the 0x80 bit, bit 1 is the 0x40 bit, and bit 7 ix the 0x01 bit.

1.4 Relationship to Protocols and Other Structures

This specification defines structures used by more than one of the ROPs as specified in [MS-OXCROPS]. It also defines structures used by more than one of the **PIM** object type specifications, such as [MS-OXOMSG] and the protocols that extend it.

The descriptions and list of properties in [MS-OXPROPS] provides context for many of the structures defined in this specification.

1.5 Applicability Statement

This specification applies to communication between clients and mailbox or public folder servers via the protocol as specified in [MS-OXCRPC].

1.6 Versioning and Localization

None.

1.7 Vendor-Extensible Fields

None.

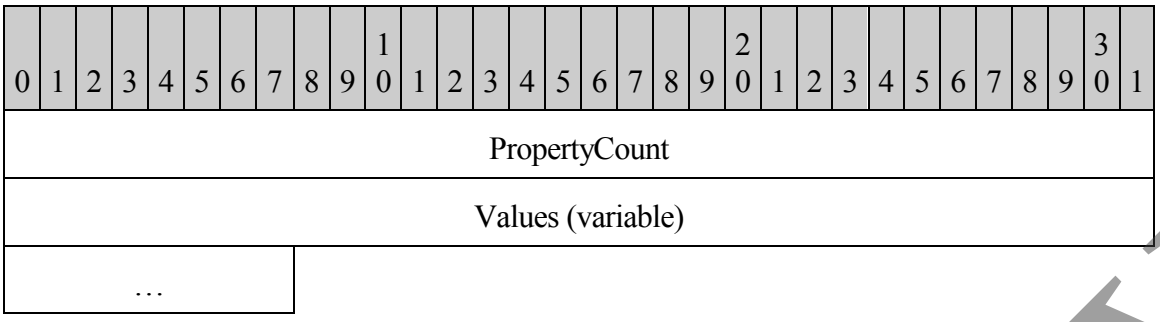
2 Structures

2.1 Address Lists

In the context of a **ROP**, addressees or recipients of a message object are represented either by a few property values or by a RecipientRow structure. In certain other contexts, such as in saved search folder criteria, addressees are represented less compactly by counted lists of property tags and values, called AddressLists.

2.1.1 AddressEntry

An AddressEntry is a set of properties representing one addressee.

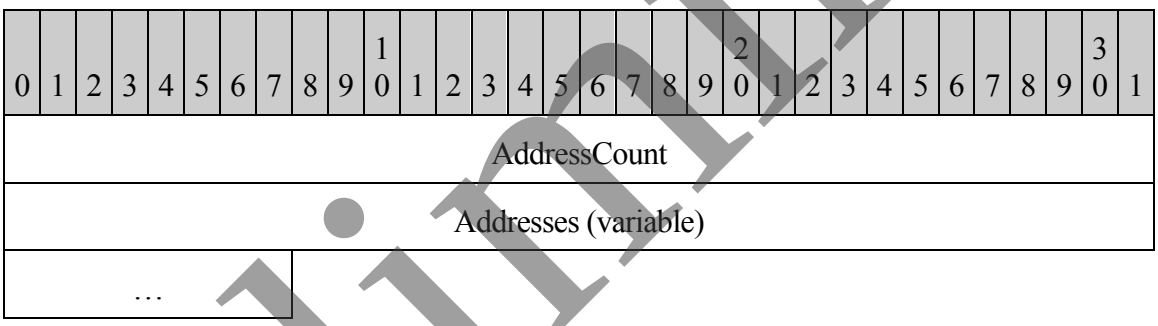


PropertyCount (4 bytes): A 32-bit unsigned integer giving the number of TaggedPropertyValues to follow. Please refer to section 0 for the specification of TaggedPropertyValue.

Values (variable): 'PropertyCount' TaggedPropertyValue structures representing one addressee.

2.1.2 AddressList

An AddressList is simply a counted set of AddressEntry structures. Each AddressEntry represents one addressee.



AddressCount (4 bytes): A 32-bit unsigned integer giving the number of addressees to follow.

Addresses (variable size): 'AddressCount' AddressEntry structures.

2.2 EntryId and Related Types

EntryId is an abstraction of an identifier for many different types of objects including folders, messages, recipients, address book entries, and message stores.

For the most common ROPs, concrete identifiers such as FolderId and MessageId – which are much more compact than EntryId – are used instead. However, there are many cases where EntryIds are stored as part or all of a binary property value, for example:

- Address book IDs are stored in a message object's PidTagSentRepresentingEntryId property

- Address book and one-off EntryIds are stored in a recipient's PidTagEntryId property
- Contact address EntryIds are stored in a contact distribution list's PidLidDistributionListMembers property

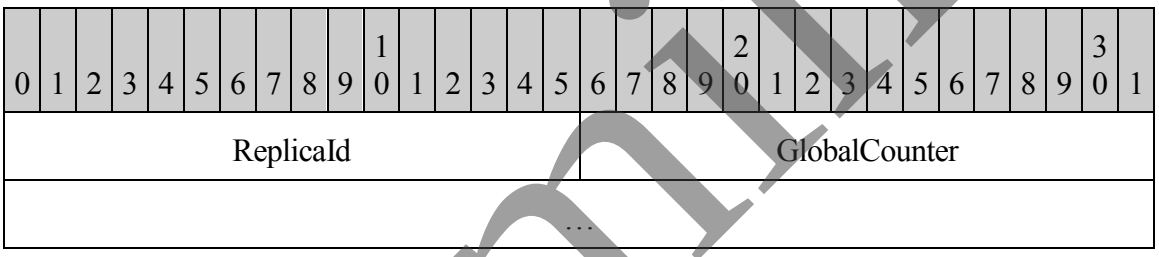
This section first describes the compact FID, MID, and GID structures, then the general EntryId structure, followed by folder, message, and message database EntryIds, and finally recipient EntryIds.

2.2.1 FID, MID, and GID

These are compact structures used in ROPs where the message database context of the objects they refer to is known.

2.2.1.1 FolderId (FID)

A FolderId uniquely identifies a folder in the context of a logon to a message database. The FolderId is serialized compactly in the context of a ROP, such as RopOpenFolder, where the message database context is already established. It is an 8-byte structure:

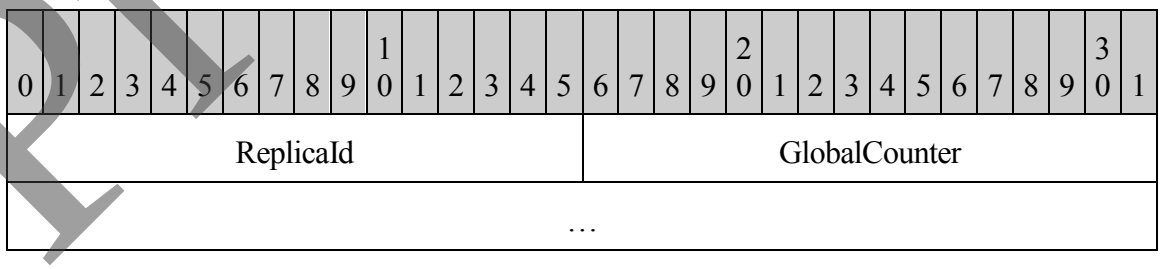


ReplicaId (2 bytes): A 16-bit unsigned integer identifying a message database.

GlobalCounter (6 bytes): An unsigned 48-bit integer identifying the folder within its message database.

2.2.1.2 MessageId (MID)

A MessageId uniquely identifies a message in the context of a logon to a message database. The MessageId is serialized compactly in the context of a ROP, such as RopOpenMessage, where the message database context is already established. It is an 8-byte structure:

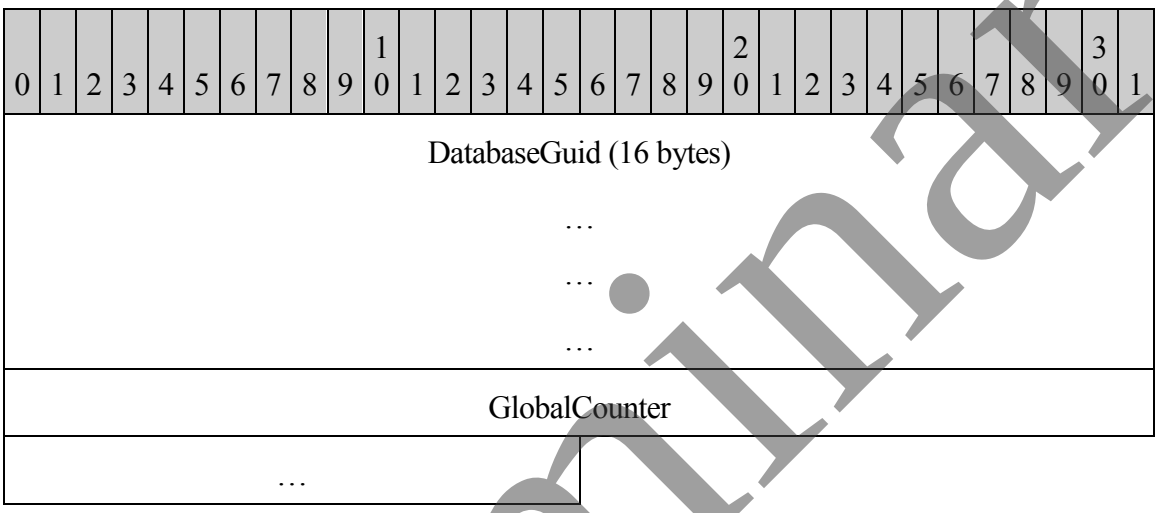


ReplicaId (2 bytes): A 16-bit unsigned integer identifying a message database.

GlobalCounter (6 bytes): An unsigned 48-bit integer identifying the message within its message database.

2.2.1.3 GID

A **GID** identifies a folder or message in a message database. It differs from a FID or MID in that the ReplicaId is replaced by the corresponding message database's GUID. The last fields of a folder or message EntryId are effectively a GID.



DatabaseGuid (16 bytes): A 128-bit unsigned integer identifying a message database.

GlobalCounter (6 bytes): An unsigned 48-bit integer identifying the folder within its message database.

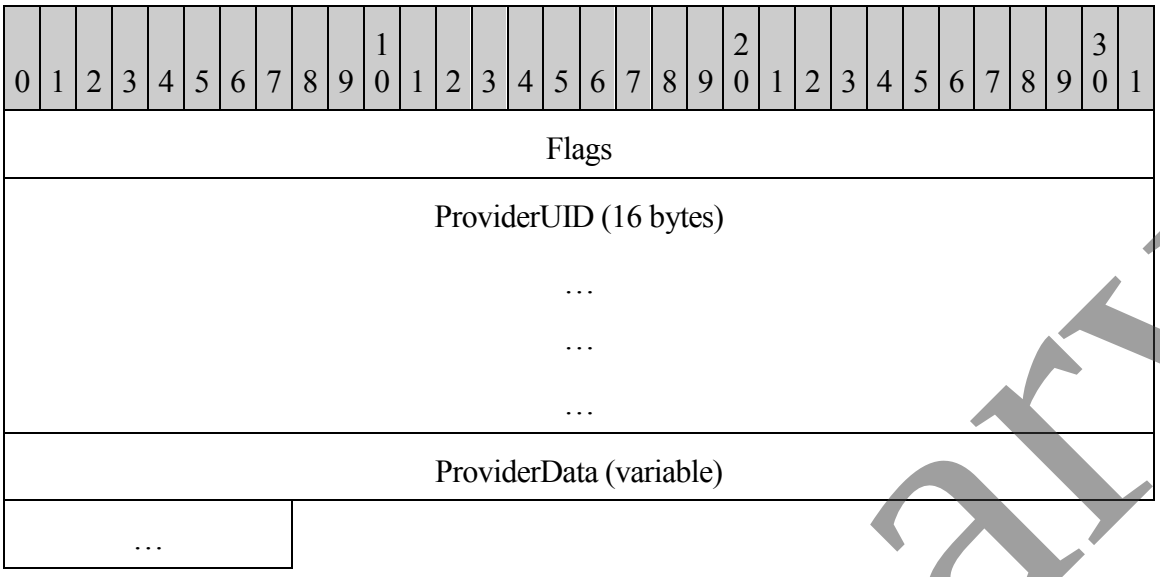
2.2.1.3.1 Long Term EntryId Structure

A Long Term EntryId (also referred to as a **LongTermID**) is a GID, as defined in section 2.2.1.3, plus a 2-byte Pad field containing 0x0000. The total length of the Long Term EntryID is 24 bytes.

Long Term EntryIds can be generated from the MID and FID by using RopLongTermIdFromId. Going the other way, MID and FID can be generated from their Long Term EntryIds by using RopIdFromLongTermId. See [MS-OXCROPS] for the ROP specifications.

2.2.2 General EntryId Structure

An EntryId carries a sequence of bytes used to identify and access an object. Note that the length of an EntryId is specified externally, not in the structure itself.



Flags (4 bytes): MUST be set to 0x00000000. Bits in this field indicate under what circumstances a short-term EntryId is valid. However, in any EntryId stored in a property value, these 4 bytes MUST be zero indicating a long-term EntryId.

ProviderUID (16 bytes): Identifies the provider that created the EntryId, and used to route EntryIDs to the correct provider. A table of values for this field appears below at Table 1.

ProviderData (variable): Provider-specific data, further specified below for several different types.

The following table specifies possible values for the ProviderUID field.

Table 1: ProviderUID Values

EntryId UID type	ProviderUID value
object in private store	%xEE.C1.BD.78.61.11.D0.11.91.7B.00.00.00.00.00.01
object in public store	%x38.A1.BB.10.05.E5.10.1A.A1.BB.08.00.2B.2A.56.C2
Address book recipient	%xDC.A7.40.C8.C0.42.10.1A.B4.B9.08.00.2B.2F.E1.82
One-off recipient	%x81.2B.1F.A4.BE.A3.10.19.9D.6E.00.DD.01.0F.54.02
Contact address or personal distribution list recipient	%xFE.42.AA.0A.18.C7.1A.10.E8.85.0B.65.1C.24.00.00

2.2.3 Message Database Object EntryIds

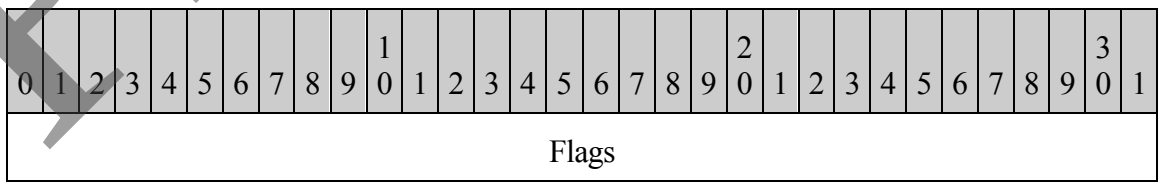
All EntryIds for objects in a message database include, at the beginning of the ProviderData field, a 16-bit unsigned integer indicating the type of object to which the EntryId corresponds. Here are the valid values for that type.

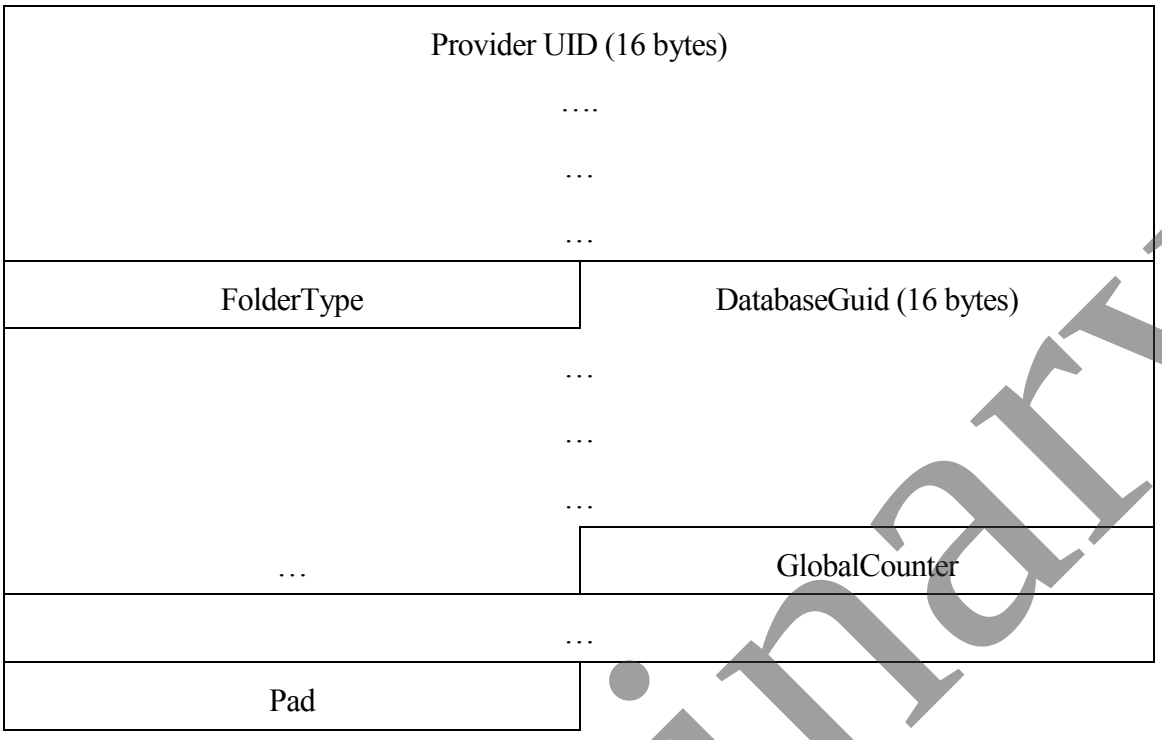
Table 2: Message database object types

Message database object type (alternate name)	Hexadecimal value
PrivateFolder (eitLTPrivateFolder)	0x0001 %x01.00
PublicFolder (eitLTPublicFolder)	0x0003 %x03.00
MappedPublicFolder (eitLTWackyFolder)	0x0005 %x05.00
PrivateMessage (eitLTPrivateMessage)	0x0007 %x07.00
PublicMessage (eitLTPublicMessage)	0x0009 %x09.00
MappedPublicMessage (eitLTWackyMessage)	0x000B %x0B.00
PublicNewsgroupFolder (eitLTPublicFolderByName)	0x000C %x0c.00

2.2.3.1 Folder EntryId

In the context of an EntryId, a FolderId looks quite different than in the context of a ROP. The ReplicaId is mapped to a DatabaseGuid; the RopLongTermIdFromId operation supports this mapping. This less compact format is necessary because no assumptions can be made about the message database context in which a folder EntryId is used.





Flags (4 bytes): MUST be zero.

Provider UID (16 bytes): MUST be %xEE.C1.BD.78.61.11.D0.11.91.7B.00.00.00.00.00.01, for a folder in a private mailbox, or %x38.A1.BB.10.05.E5.10.1A.A1.BB.08.00.2B.2A.56.C2, for a folder in the public store.

FolderType (2 bytes): One of several types as specified in Table 2 above.

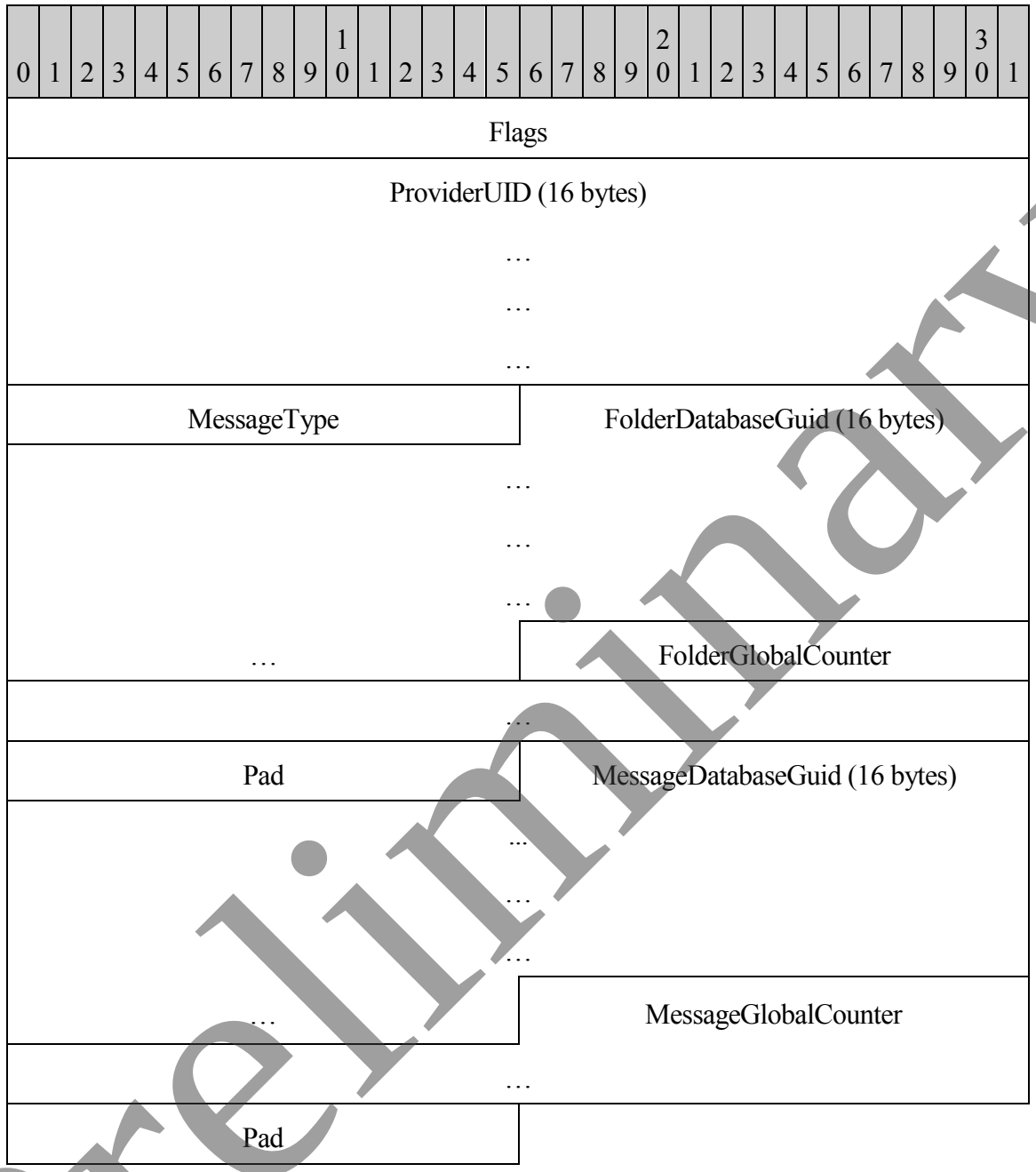
DatabaseGuid (16 bytes): A GUID associated with the message database, and corresponding to the ReplicaId field of the FID.

GlobalCounter (6 bytes): An unsigned 48-bit integer identifying the folder.

Pad (2 bytes): MUST be zero.

2.2.3.2 Message EntryId

In the context of an EntryId, a MessageId looks quite different than in the context of a ROP. The DatabaseReplicationId is mapped to a MessageDatabaseGuid (perhaps using the RopLongTermIdFromId operation) and the whole ID is prefixed with flags and a provider UID. In addition, the FolderId of the folder in which the message resides is included.



Flags (4 bytes): MUST be 0x00000000.

ProviderUID (16 bytes): MUST be either %xEE.C1.BD.78.61.11.D0.11.91.7B.00.00.00.00.00.01, for a folder in a private mailbox, or %x38.A1.BB.10.05.E5.10.1A.A1.BB.08.00.2B.2A.56.C2, for a folder in the public store.

MessageType (2 bytes): One of several types as specified in Table 2 above.

FolderDatabaseGuid (16 bytes): A GUID associated with the message database of the folder in which the message resides, and corresponding to the DatabaseReplicationId field of the FolderId.

FolderGlobalCounter (6 bytes): An unsigned 48-bit integer identifying the folder in which the message resides.

Pad (2 bytes): MUST be zero.

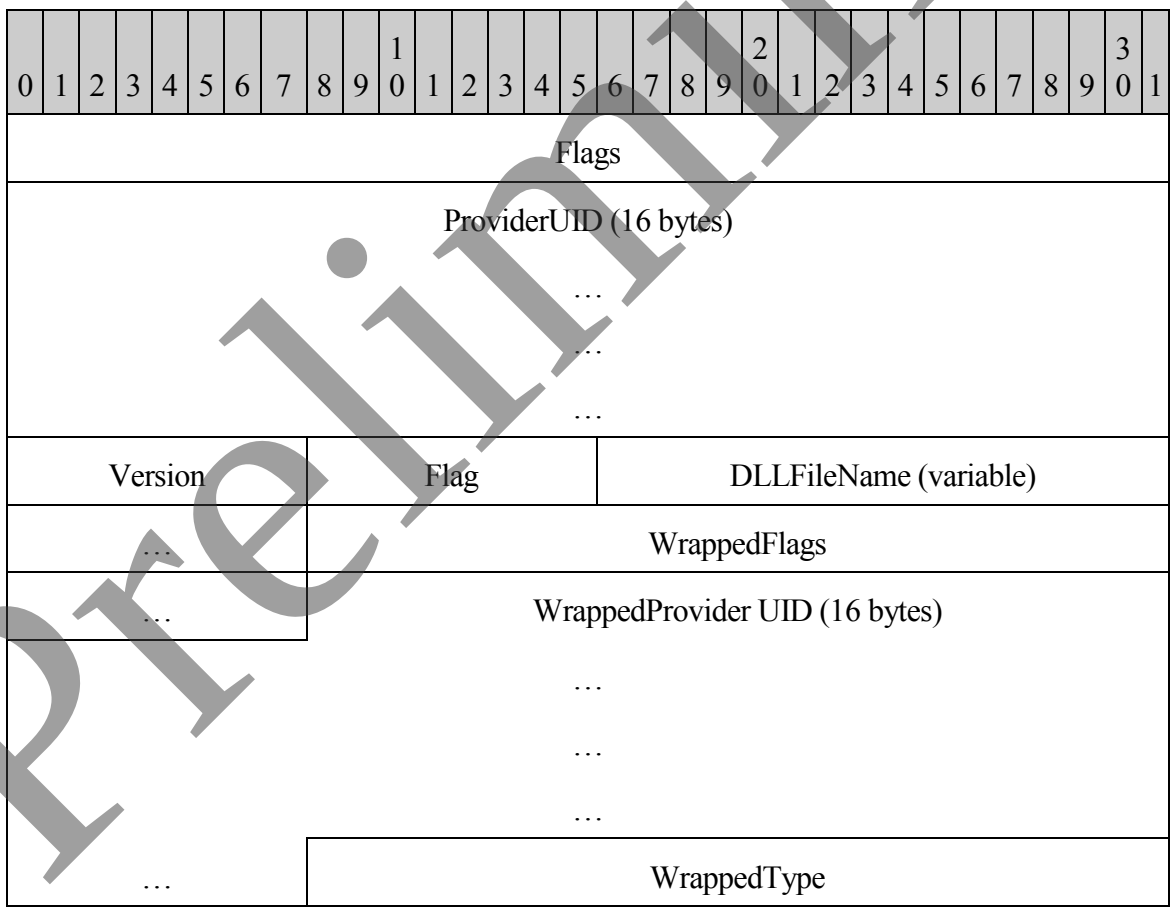
MessageDatabaseGuid (16 bytes): A GUID associated with the message database of the message and corresponding to the DatabaseReplicationId field of the MessageId.

MessageGlobalCounter (6 bytes): An unsigned 48-bit integer identifying the message.

Pad (2 bytes): MUST be zero.

2.2.3.3 Message Database EntryIds

A message database EntryId identifies a mailbox message database or a public folder message database itself, rather than a message or folder object residing in such a database. It is used in certain property values.



…	ServerShortname (variable)
…	MailboxDN (variable)
…	

Flags (4 bytes): MUST be 0x00000000.

ProviderUID (16 bytes): MUST be either %xEE.C1.BD.78.61.11.D0.11.91.7B.00.00.00.00.00.01, for a folder in a private mailbox, or %x38.A1.BB.10.05.E5.10.1A.A1.BB.08.00.2B.2A.56.C2, for a folder in the public store.

Version (1 byte): MUST be zero.

Flag (1 byte): MUST be zero.

DLLFileName (variable): MUST be set to the following value which represents "emsmdb.dll": %x45.4D.53.4D.44.42.2E.44.4C.4C.00.00.00.00

WrappedFlags (4 bytes): MUST be 0x00000000.

WrappedProvider UID (16 bytes): MUST be one of the following values:

Message database type	ProviderUID value
Mailbox message database	%x1B.55.FA.20.AA.66.11.CD.9B.C8.00.AA.00.2F.C4.5A
Public folder message database	%x1C.83.02.10.AA.66.11.CD.9B.C8.00.AA.00.2F.C4.5A

WrappedType (4 bytes): MUST be %x0C.00.00.00 for a mailbox store, or %x06.00.00.00 for a public store.

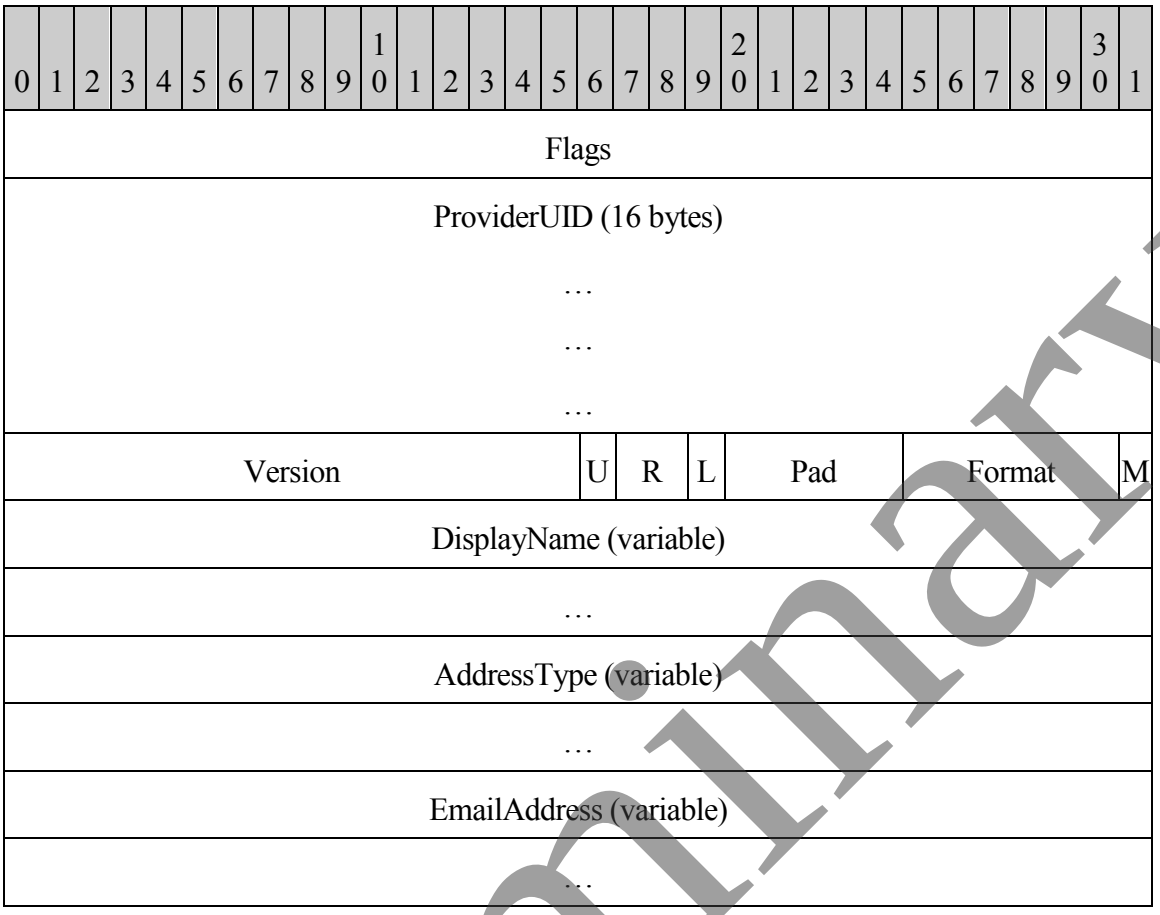
ServerShortname (variable): A string of single-byte characters terminated by a single zero byte, indicating the shortname or NetBIOS name of the server.

MailboxDN (variable): A string of single-byte characters terminated by a single zero byte and representing the **X500 DN** of the mailbox, as specified in [MS-OXOAB]. This field is present only for mailbox databases.

2.2.4 Recipient EntryIds

2.2.4.1 One-Off EntryId

One-off EntryIds are used to hold information about recipients that do not exist in the directory. All information about a one-off recipient is contained in the EntryId itself.



Flags (4 bytes): MUST be 0x00000000.

ProviderUID (16 bytes): MUST be %x81.2B.1F.A4.BE.A3.10.19.9D.6E.00.DD.01.0F.54.02.

Version (2 bytes): MUST be 0x0000.

U (1 bit): 1-bit flag (0x8000). If 1, the string fields following are in Unicode (UTF-16) with two-byte null terminators; if 0, the string fields following are MBCS characters terminated by a single 0 byte.

R (2 bits): Reserved (mask 0x6000), MUST be zero.

L (1 bit): 1-bit flag (0x1000). If 1, server SHOULD NOT attempt to look up this user's e-mail address in the address book.

Pad (5 bits): Reserved (mask 0x0F80), MUST be 0.

Format (6 bits): (6-bit enumeration, mask 0x007E) The message format desired for this recipient, as specified in the following table.

Table 3: One-Off EntryId Formatting Values

Name	Word value	Field value	Description
TextOnly	0x0006	b’000011’	Send a plain text message body.
HtmlOnly	0x000E	b‘000111’	Send an HTML message body.
TextAndHtml	0x0016	b’001011’	Send a multipart/alternative body with both plain text and HTML.

M (1 bit): 1-bit flag (0x0001**).** If 0, recipient SHOULD receive messages in TNEF format; if 1, recipient SHOULD receive messages in MIME format.

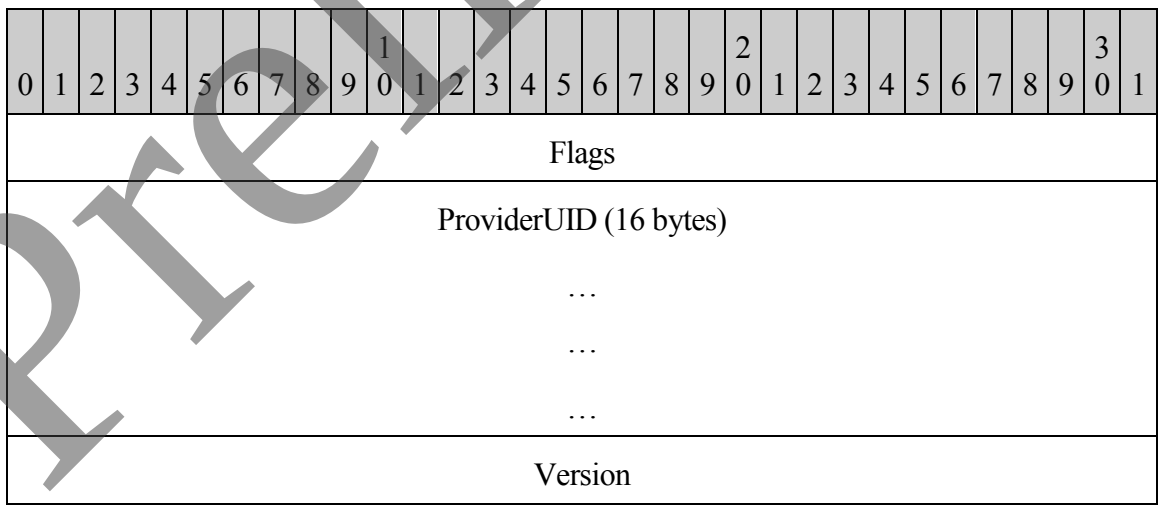
DisplayName (variable): The recipient’s display name (in the recipient table, PidTagDisplayName) as a null-terminated string. If the U field is 1, the null terminator is 2 bytes long; otherwise, 1 byte.

AddressType (variable): The recipient’s e-mail address type (in the recipient table, PidTagAddressType) as a null-terminated string. If the U field is 1, the null terminator is 2 bytes long; otherwise, 1 byte.

EmailAddress (variable): The recipient’s e-mail address (in the recipient table, PidTagEmailAddress) as a null-terminated string. If the U field is 1, the null terminator is 2 bytes long; otherwise, 1 byte.

2.2.4.2 Address Book EntryId

Address book EntryIds can represent several types of address book objects including individual users, distribution lists, containers, and templates as specified in Table 4.



Type
X500DN (variable)
…

Flags (4 bytes): MUST be 0x00000000.

ProviderUID (16 bytes): MUST be %xDC.A7.40.C8.C0.42.10.1A.B4.B9.08.00.2B.2F.E1.82. (Directory)

Version (4 bytes): MUST be set to %x01.00.00.00.

Type (4 bytes): A 32-bit integer representing the type of the object. It MUST be one of the values from the following table. For more information, see [MS-OXABK].

Table 4: Address Book Object Types

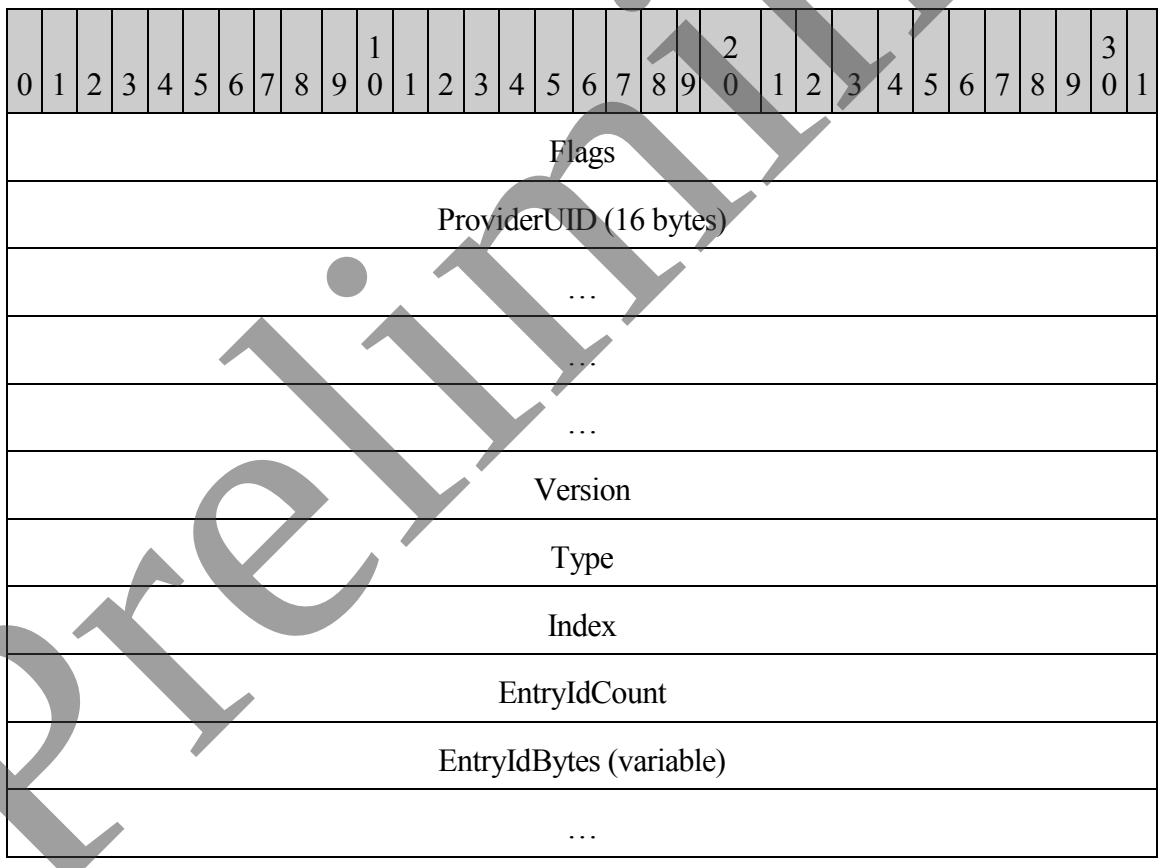
Value (hex bytes)	Address book EntryId type
0x00000000 %x00.00.00.00	Local mail user
0x00000001 %x01.00.00.00	Distribution list
0x00000002 %x02.00.00.00	Bulletin board or public folder
0x00000003 %x03.00.00.00	Automated mailbox
0x00000004 %x04.00.00.00	Organizational mailbox
0x00000005 %x05.00.00.00	Private distribution list
0x00000006 %x06.00.00.00	Remote mail user
0x00000100 %x00.01.00.00	Container

0x00000101 %x01.01.00.00	Template
0x00000102 %x02.01.00.00	One-off user
0x00000200 %x00.02.00.00	Search

X500DN (variable): The X500 DN of the address book object. X500DN is a null-terminated string of 8-bit characters.

2.2.4.3 Contact Address EntryId

Contact Address EntryIds represent recipients whose information is stored in a Contact object, as specified in [MS-OXCNTC].



Flags (4 bytes): MUST be %x00.00.00.00.

ProviderUID (16 bytes): MUST be %xFE.42.AA.0A.18.C7.1A.10.E8.85.0B.65.1C.24.00.00.

Version (4 bytes): MUST be %x03.00.00.00.

Type (4 bytes): MUST be %x04.00.00.00.

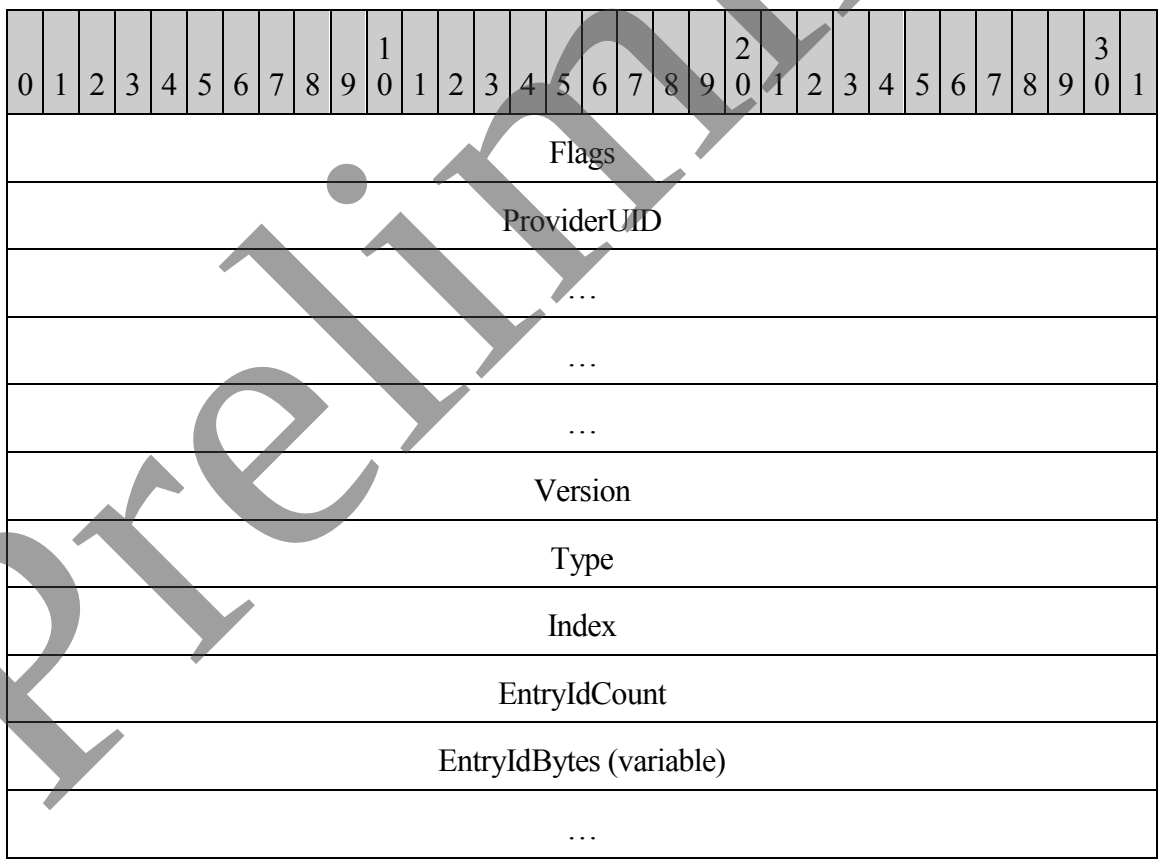
Index (4 bytes): 4-byte unsigned integer value. This value MUST be a number between 0 and 5 (inclusive) and represents which electronic address in the contact information to use. A value of 0, 1, and 2 represents Email1, Email2, Email3 respectively, and a value of 3, 4, and 5 represents Fax1, Fax2 and Fax3 respectively. For more information, see [MS-OXCNTC].

EntryIdCount (4 bytes): 4-byte unsigned integer value representing the count of bytes in the EntryIdBytes field.

EntryIdBytes (variable): EntryId of the Contact object that contains this address, which in turn has a format specified in section 2.2.3.2. The size of this structure is specified by the EntryIdCount field <1>.

2.2.4.4 Personal Distribution List EntryId

The Personal Distribution List **entry IDs** represents recipients whose information is stored in a Personal Distribution List object, as specified in [MS-OXCNTC].



Flags (4 bytes): MUST be %x00.00.00.00.

ProviderUID (16 bytes): MUST be %xFE.42.AA.0A.18.C7.1A.10.E8.85.0B.65.1C.24.00.00.

Version (4 bytes): MUST be %x03.00.00.00.

Type (4 bytes): MUST be %x05.00.00.00.

Index (4 bytes): MUST be %xFF.00.00.00.

EntryIdCount (4 bytes): 4-byte unsigned integer value representing the count of bytes in the EntryIdBytes field.

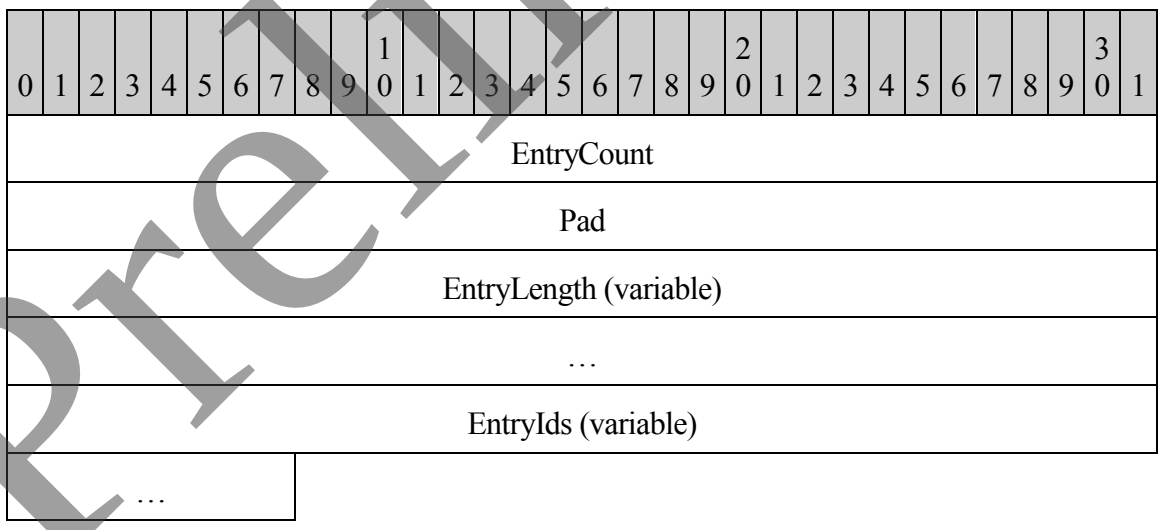
EntryIdBytes (variable): EntryId of the Personal Distribution List object to which this address refers, which in turn has the format specified in section 2.2.3.2. The size of this structure is specified by the EntryIdCount field <2>.

2.3 EntryId Lists

2.3.1 EntryList

EntryList is used in search folder criteria to serialize a list of EntryIds. Briefly, there are three parts to this structure:

- The count of entries in the list
- 'count' structures giving the length of individual entries
- Data for each of the 'count' individual entries



EntryCount (4 bytes): An unsigned 32-bit integer giving the number of EntryIds in the list. It MUST be followed by that many EntryLength and that many EntryId structures.

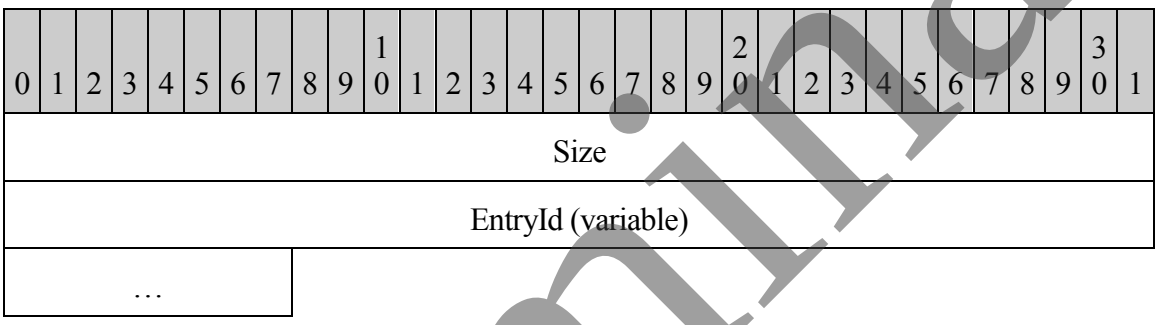
Pad (4 bytes): Clients MAY store any value; servers MUST ignore the value.

EntryLength (EntryCount * 8 bytes): A series of 'EntryCount' pairs: an unsigned 32-bit integer giving the size of one EntryId, followed by 4-byte pad that MAY have any value.

EntryIds (variable size): A series of 'EntryCount' EntryIds. There is no padding between EntryIds. The length of the i'th EntryId is specified by the first 32 bits of the i'th element of the EntryLength.

2.3.2 FlatEntry

A FlatEntry structure is simply the size of an EntryId, followed by the EntryId itself, for ease of serialization.

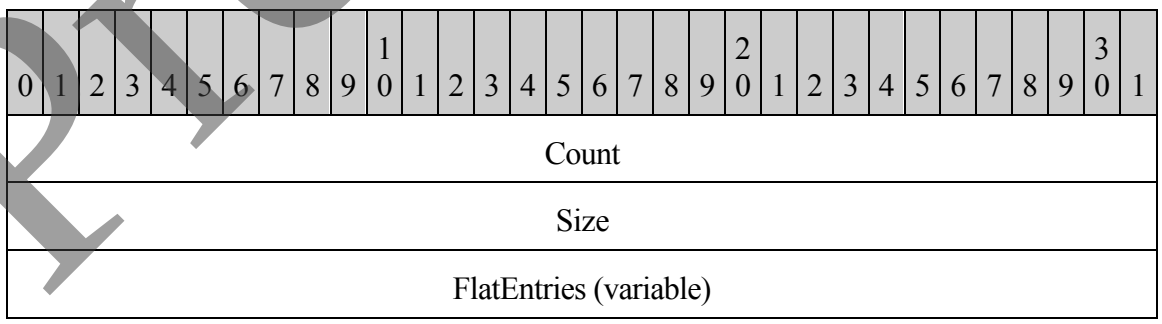


Size (4 bytes): An unsigned 32-bit integer giving the size of the following EntryId, not including the Size field itself.

EntryId: The EntryId itself. It MUST be exactly Size bytes long.

2.3.3 FlatEntryList

A FlatEntryList gives the number of EntryIds and their total size, followed by a series of FlatEntry structures.



…

Count (4 bytes): An unsigned 32-bit integer giving the number of FlatEntry structures in the list.

Size (4 bytes): The total size of all the FlatEntry structures, not including the Count and Size fields themselves.

FlatEntries (variable size): A series of FlatEntry structures with the actual EntryId data. There MUST be exactly Count structures, and their total size MUST be exactly Size.

2.4 Error Codes

When encoded in ROP buffers, all error codes are transmitted as 32-bit integers in little-endian format. Error codes are presented in the following table:.

Table 5 Error Codes

Name	Description (alternate names)	Numeric value (hex)
Success	The operation succeeded. (S_OK, SUCCESS_SUCCESS)	00000000, %x00.00.00.00
GeneralFailure	The operation failed for an unspecified reason (E_FAIL, MAPI_E_CALL_FAILED, ecError, SYNC_E_ERROR)	80004005, %x05.40.00.80
OutOfMemory	Not enough memory was available to complete the operation (E_NOMEMORY, MAPI_E_NOT_ENOUGH_MEMORY, ecMAPIOOM, ecPropSize)	8007000E, %x0E.00.07.80
InvalidParameter	An invalid parameter was passed to a remote procedure call (E_INVALIDARG, MAPI_E_INVALID_PARAMETER, ecInvalidParam, ecInvalidSession, ecBadBuffer, SYNC_E_INVALID_PARAMETER)	80070057, %x57.00.07.80
NoInterface	The requested interface is not supported (E_NOINTERFACE, MAPI_E_INTERFACE_NOT_SUPPORTED, ecinterfacenotsupported)	80004002 %x02.40.00.80

Name	Description (alternate names)	Numeric value (hex)
AccessDenied	The caller does not have sufficient access rights to perform the operation (E_ACCESSDENIED, MAPI_E_NO_ACCESS, ecaccessdenied, ecpropsecurityviolation)	80070005, %x05.00.07.80
NotSupported	The server does not support this method call. (MAPI_E_NO_SUPPORT, ecNotSupported, ecNotImplemented)	80040102, %x02.01.04.80
InvalidCharacterWidth	Unicode characters were requested when only 8-bit characters are supported, or vice versa. (MAPI_E_BAD_CHARWIDTH, ecBadCharwidth)	80040103, %x03.01.04.80
StringTooLong	In the context of this method call, a string exceeds the maximum permitted length. (MAPI_E_STRING_TOO_LONG, ecStringTooLarge)	80040105, %x05.01.04.80
InvalidFlag	An unrecognized flag bit was passed to a method call. (MAPI_E_UNKNOWN_FLAGS, ecUnknownFlags, SYNC_E_UNKNOWN_FLAGS)	80040106, %x06.01.04.80
InvalidEntryID	An incorrectly formatted EntryId was passed to a method call. (MAPI_E_INVALID_ENTRYID, ecInvalidEntryId)	80040107, %x07.01.04.80
InvalidObject	A method call was made using a reference to an object that has been destroyed or is not in a viable state. (MAPI_E_INVALID_OBJECT, ecInvalidObject)	80040108, %x08.01.04.80

Name	Description (alternate names)	Numeric value (hex)
ObjectChanged	An attempt to commit changes failed because the object was changed separately. (MAPI_E_OBJECT_CHANGED, ecObjectModified)	80040109, %x09.01.04.80
ObjectDeleted	An operation failed because the object was deleted separately. (MAPI_E_OBJECT_DELETED, ecObjectDeleted)	8004010A, %x0A.01.04.80
ServerBusy	A table operation failed because a separate operation was in progress at the same time. (MAPI_E_BUSY, ecBusy)	8004010B, %x0B.01.04.80
OutOfDisk	Not enough disk space was available to complete the operation. (MAPI_E_NOT_ENOUGH_DISK, ecDiskFull)	8004010D, %x0D.01.04.80
OutOfResources	Not enough of an unspecified resource was available to complete the operation. (MAPI_E_NOT_ENOUGH_RESOURCES, ecInsufficientResrc)	8004010E, %x0E.01.04.80
NotFound	The requested object could not be found at the server. (MAPI_E_NOT_FOUND, ecNotFound, ecAttachNotFound, ecUnknownRecip, ecPropNotExistent)	8004010F, %x0F.01.04.80
VersionMismatch	Client and server versions are not compatible. (MAPI_E_VERSION, ecVersionMismatch, ecVersion)	80040110, %x10.01.04.80
LogonFailed	A client was unable to log on to the server. (MAPI_E_LOGON_FAILED, ecLoginFailure)	80040111, %x11.01.04.80

Name	Description (alternate names)	Numeric value (hex)
TooManySessions	A server or service is unable to create any more sessions. (MAPI_E_SESSION_LIMIT, ecTooManySessions)	80040112, %x12.01.04.80
UserCanceled	An operation failed because a user cancelled it. (MAPI_E_USER_CANCEL, ecUserAbort)	80040113, %x13.01.04.80
AbortFailed	A RopAbort or RopAbortSubmit request was unsuccessful. (MAPI_E_UNABLE_TO_ABORT, ecUnableToAbort)	80040114, %x14.01.04.80
NetworkError	An operation was unsuccessful because of a problem with network operations or services. (MAPI_E_NETWORK_ERROR, ecNetwork)	80040115, %x15.01.04.80
DiskError	There was a problem writing to or reading from disk. (MAPI_E_DISK_ERROR, ecWriteFault, ecReadFault)	80040116, %x16.01.04.80
TooComplex	The operation requested is too complex for the server to handle; often applied to restrictions. (MAPI_E_TOO_COMPLEX, ecTooComplex)	80040117, %x17.01.04.80
InvalidColumn	The column requested is not allowed in this type of table. (MAPI_E_BAD_COLUMN)	80040118, %x18.01.04.80
ComputedValue	A property cannot be updated because it is read-only, computed by the server. (MAPI_E_COMPUTED, ecComputed)	8004011A, %x1A.01.04.80

Name	Description (alternate names)	Numeric value (hex)
CorruptData	There is an internal inconsistency in a database, or in a complex property value. (MAPI_E_CORRUPT_DATA, ecCorruptData)	8004011B, %x1B.01.04.80
InvalidCodepage	The server is not configured to support the codepage requested by the client. (MAPI_E_UNKNOWN_CPID)	8004011E, %x1E.01.04.80
InvalidLocale	The server is not configured to support the locale requested by the client. (MAPI_E_UNKNOWN_LCID)	8004011F, %x1F.01.04.80
TimeSkew	The operation failed due to clock skew between servers. (MAPI_E_INVALID_ACCESS_TIME, ecTimeSkew)	80040123, %x23.01.04.80
EndOfSession	Indicates that the server session has been destroyed, possibly by a server restart, and the client SHOULD reconnect. (MAPI_E_END_OF_SESSION)	80040200, %x00.02.04.80
UnknownEntryId	Indicates that the EntryId passed to OpenEntry was created by a different MAPI provider. (MAPI_E_UNKNOWN_ENTRYID)	80040201, %x01.02.04.80
NotCompleted	A complex operation such as building a table row set could not be completed. (MAPI_E_UNABLE_TO_COMPLETE, ecUnableToComplete)	80040400, %x00.04.04.80
Timeout	An asynchronous operation did not succeed within the specified timeout. (MAPI_E_TIMEOUT, ecTimeout)	80040401, %x01.04.04.80
EmptyTable	A table essential to the operation is empty. (MAPI_E_TABLE_EMPTY, ecTableEmpty)	80040402, %x02.04.04.80

Name	Description (alternate names)	Numeric value (hex)
TableTooBig	The table is too big for the requested operation to complete. (MAPI_E_TABLE_TOO_BIG, ecTableTooBig)	80040403, %x03.04.04.80
InvalidBookmark	The bookmark passed to a table operation was not created on the same table. (MAPI_E_INVALID_BOOKMARK, ecInvalidBookmark)	80040405, %x05.04.04.80
ErrorWait	A wait timeout has expired. (MAPI_E_WAIT, ecWait)	80040500, %x00.05.04.80
ErrorCancel	The operation had to be canceled, (MAPI_E_CANCEL, ecCancel)	80040501, %x01.05.04.80
NoSuppress	The server does not support the suppression of read receipts. (MAPI_E_NO_SUPPRESS)	80040602, %x02.06.04.80
CollidingNames	A folder or item cannot be created because one with the same name or other criteria already exists. (MAPI_E_COLLISION, ecDuplicateName)	80040604, %x04.06.04.80
NotInitialized	The subsystem is not ready. (MAPI_E_NOT_INITIALIZED, ecNotInitialized)	80040605, %x05.06.04.80
NoRecipients	A message cannot be sent because it has no recipients. (MAPI_E_NO_RECIPIENTS)	80040607, %x07.06.04.80
AlreadySent	A message cannot be opened for modification because it has already been sent. (MAPI_E_SUBMITTED, ecSubmitted)	80040608, %x08.06.04.80

Name	Description (alternate names)	Numeric value (hex)
HasFolders	A folder cannot be deleted because it still contains subfolders. (MAPI_E_HAS_FOLDERS, ecFolderHasChildren)	80040609, %x09.06.04.80
HasMessages	A folder cannot be deleted because it still contains messages. (MAPI_E_HAS_MESSAGES, ecFolderHasContents)	8004060A, %x0A.06.04.80
FolderCycle	A folder move or copy operation would create a cycle (typically when the request is to copy a parent folder to one of its subfolders). (MAPI_E_FOLDER_CYCLE, ecRootFolder)	8004060B, %x0B.06.04.80
TooManyLocks	Too many locks have been requested. (MAPI_E_LOCKID_LIMIT, ecLockIdLimit)	8004060D, %x0D.06.04.80
AmbiguousRecipient	An unresolved recipient matches more than one entry in the directory. (MAPI_E_AMBIGUOUS_RECIP, ecAmbiguousRecip)	80040700, %x00.07.04.80
ObjectDeleted	The requested object was previously deleted. (SYNC_E_OBJECT_DELETED)	80040800, %x00.08.04.80
IgnoreFailure	An error occurred but it's safe to ignore the error, perhaps because the change in question has been superseded. (SYNC_E_IGNORE)	80040801 %x01.08.04.80
SyncConflict	Conflicting changes to an object have been detected. (SYNC_E_CONFLICT)	80040802 %x02.08.04.80
NoParentFolder	The parent folder could not be found. (SYNC_E_NO_PARENT)	80040803 %x03.08.04.80

Name	Description (alternate names)	Numeric value (hex)
CycleDetected	An operation would create a cycle (for instance, by copying a parent folder to one of its subfolders).	80040804 %x04.08.04.80
NotSynchronized	A sync operation did not take place, possibly due to a conflicting change. (SYNC_E_UNSYNCHRONIZED)	80040805 %x05.08.04.80
NamedPropertyQuota	The message database cannot store any more named property mappings. (MAPI_E_NAMED_PROP_QUOTA_EXCEEDED, ecNPQuotaExceeded)	80040900, %x00.09.04.80

2.4.1 Additional Error Codes

When encoded in ROP buffers, all error codes are transmitted as 32-bit integers in little-endian format. Additional error codes are presented in the following table:

Table 6 Additional Error Codes

Name	Description (alternate names)	Numeric value (hex)
JetError	Unspecified database failure. (ecJetError)	0x000003EA
UnknownUser	Unable to identify a home message database for this user. (ecUnknownUser)	0x000003EB
Exiting	The server is in the process of stopping. (ecExiting)	0x000003ED
BadConfiguration	Protocol settings for this user are incorrect. (ecBadConfig)	0x000003EE
UnknownCodePage	The specified codepage is not installed on the server. (ecUnknownCodePage)	0x000003EF

ServerMemory	The server is out of memory. (ecServerOOM, ecMemory)	0x000003F0
LoginPermission	This user does not have access rights to the mailbox. (ecLoginPerm)	0x000003F2
DatabaseRolledBack	The database has been restored and needs fixup, but cannot be fixed up. (ecDatabaseRolledBack)	0x000003F3
DatabaseCopiedError	The database file has been copied from another server. (ecDatabaseCopiedError)	0x000003F4
AuditNotAllowed	Auditing of security operations is not permitted. (ecAuditNotAllowed)	0x000003F5
ZombieUser	User has no security identifier. (ecZombieUser)	0x000003F6
UnconvertableACL	An access control list cannot be converted to NTFS format. (ecUnconvertableACL)	0x000003F7
NoFreeJetSessions	No Jet session is available. (ecNoFreeJses)	0x0000044C
DifferentJetSession	Warning, a Jet session other than the one requested was returned. (ecDifferentJses)	0x0000044D
FileRemove	An error occurred when attempting to remove a database file. (ecFileRemove)	0x0000044F
ParameterOverflow	Parameter value overflow. (ecParameterOverflow)	0x00000450
BadVersion	Bad message store database version number. (ecBadVersion)	0x00000451
TooManyColumns	Too many columns requested in SetColumns. (ecTooManyCols)	0x00000452

HaveMore	A ROP has more data to return. (ecHaveMore)	0x00000453
DatabaseError	General database problem (ecDatabaseError)	0x00000454
IndexNameTooBig	An index name is larger than what Jet allows (ecIndexNameTooBig)	0x00000455
UnsupportedProperty	The property data type is not supported. (ecUnsupportedProp)	0x00000456
MessageNotSaved	During AbortSubmit, a message was not saved. (ecMsgNotSaved)	0x00000457
UnpublishedNotification	A notification could not be published at this time. (ecUnpubNotif)	0x00000459
DifferentRoot	Moving or copying folders to a different top-level hierarchy is not supported. (ecDifferentRoot)	0x0000045B
BadFolderName	Invalid folder name. (ecBadFolderName)	0x0000045C
AttachmentOpen	The attachment is open. (ecAttachOpen)	0x0000045D
InvalidCollapseState	The collapse state given to SetCollapseState is invalid. (ecInvClpsState)	0x0000045E
SkipMyChildren	While walking a folder tree, do not consider children of this folder. (ecSkipMyChildren)	0x0000045F
SearchFolder	The operation is not supported on a search folder. (ecSearchFolder)	0x00000460
NotSearchFolder	The operation is valid only on a search folder. (ecNotSearchFolder)	0x00000461

FolderSetReceive	This is a receive folder and cannot be deleted. (ecFolderSetReceive)	0x00000462
NoReceiveFolder	No receive folder is available (even no default). (ecNoReceiveFolder)	0x00000463
DeleteSubmittedMessage	Deleting a message that has been submitted for sending is not permitted. (ecNoDelSubmitMsg)	0x00000465
InvalidRecipients	It was impossible to deliver to this recipient. (ecInvalidRecips)	0x00000467
NoReplicaHere	No replica of the public folder in this message database. (ecNoReplicaHere)	0x00000468
NoReplicaAvailable	No available message database has a replica of this public folder. (ecNoReplicaAvailable)	0x00000469
PublicDatabase	The operation is invalid on a public message database. (ecPublicMDB)	0x0000046A
NotPublicDatabase	The operation is valid only on a public message database. (ecNotPublicMDB)	0x0000046B
RecordNotFound	The record was not found. (ecRecordNotFound)	0x0000046C
ReplicationConflict	A replication conflict was detected. (ecReplConflict)	0x0000046D
FXBufferOverrun	Prevented an overrun while reading a fast transfer buffer. (ecFxBufferOverrun)	0x00000470
FXBufferEmpty	No more in a fast transfer buffer. (ecFxBufferEmpty)	0x00000471
FXPartialValue	Partial long value in a fast transfer buffer. (ecFxPartialValue)	0x00000472

FxNoRoom	No room for an atomic value in a fast transfer buffer. (ecFxNoRoom)	0x00000473
TimeExpired	Housekeeping functions have exceeded their time window. (ecMaxTimeExpired)	0x00000474
DestinationError	An error occurred on the destination folder during a copy operation. (ecDstError)	0x00000475
DatabaseNotInitialized	The message database was not properly initialized. (ecMDBNotInit)	0x00000476
WrongServer	This server does not host the user's mailbox database. (ecWrongServer)	0x00000478
BufferTooSmall	A buffer passed to this function is not big enough. (ecBufferTooSmall)	0x0000047D
AttachmentResolutionRequired	Linked attachments could not be resolved to actual files. (ecRequiresRefResolve)	0x0000047E
ServerPaused	The service is in a paused state. (ecServerPaused)	0x0000047F
ServerBusy	The server is too busy to complete an operation. (ecServerBusy)	0x00000480
NoSuchLogon	No such logon exists in the message database's Logon list. (ecNoSuchLogon)	0x00000481
LoadLibraryFailed	Internal error: the service cannot load a required DLL. (ecLoadLibFailed)	0x00000482

AlreadyConfigured	A synchronization object has already been configured. (ecObjAlreadyConfig)	0x00000483
NotConfigured	A synchronization object has not yet been configured. (ecObjNotConfig)	0x00000484
DataLoss	A CodePage conversion incurred when data loss. (ecDataLoss)	0x00000485
MaximumSendThreadExceeded	The maximum number of send threads has been exceeded. (ecMaxSendThreadExceeded)	0x00000488
FxErrorMarker	A fast transfer error marker was found, and recovery is necessary. (ecFxErrorMarker)	0x00000489
NoFreeJtabs	There are no more free Jet tables. (ecNoFreeJtabs)	0x0000048A
NotPrivateDatabase	The operation is only valid on a private mailbox database. (ecNotPrivateMDB)	0x0000048B
IsintegMDB	The message database has been locked by the ISINTEG utility. (ecIsintegMDB)	0x0000048C
RecoveryMismatch	A recovery storage group operation was attempted on a non-RSG message database, or vice-versa. (ecRecoveryMDBMismatch)	0x0000048D
TableMayNotBeDeleted	Attempt to delete a critical table, such as the Messages or Attachments table. (ecTableMayNotBeDeleted)	0x0000048E
RpcRegisterIf	Error in registering RPC interfaces. (ecRpcRegisterIf)	0x000004B1
RpcListen	Error in starting the RPC listener. (ecRpcListen)	0x000004B2

RpcFormat	A badly formatted RPC buffer was detected. (ecRpcFormat)	0x000004B6
NoCopyTo	Single instance storage cannot be used in this case. (ecNoCopyTo)	0x000004B7
NullObject	An object handle reference in the RPC buffer could not be resolved. (ecNullObject)	0x000004B9
RpcAuthentication	Server requests client to use authentication. (ecRpcAuthentication)	0x000004BC
RpcBadAuthenticationLevel	The server doesn't recognize a client's authentication level. (ecRpcBadAuthenticationLevel)	0x000004BD
NullCommentRestriction	The sub-restriction of a comment restriction is empty. (ecNullCommentRestriction)	0x000004BE
RulesLoadError	Rule data was unavailable for this folder. (ecRulesLoadError)	0x000004CC
RulesDeliverErr	Delivery-time failure in rule execution. (ecRulesDeliverErr)	0x000004CD
RulesParsingErr	Invalid syntax in a stored rule condition or action. (ecRulesParsingErr)	0x000004CE
RulesCreateDAE	Failure creating a deferred rule action error message. (ecRulesCreateDaeErr)	0x000004CF
RulesCreateDAM	Failure creating a deferred rule action message. (ecRulesCreateDamErr)	0x000004D0
RulesNoMoveCopyFolder	A move or copy rule action could not be performed due to a problem with the target folder. (ecRulesNoMoveCopyFolder)	0x000004D1

RulesNoFolderRights	A move or copy rule action could not be performed due to a permissions problem with the target folder. (ecRulesNoFolderRights)	0x000004D2
MessageTooBig	A message could not be delivered because it exceeds a size limit. (ecMessageTooBig)	0x000004D4
FormNotValid	There is a problem with the form mapped to the message's message class. (ecFormNotValid)	0x000004D5
NotAuthorized	Delivery to the desired folder was not authorized. (ecNotAuthorized)	0x000004D6
DeleteMessage	The message was deleted by a rule action. (ecDeleteMessage)	0x000004D7
BounceMessage	Delivery of the message was denied by a rule action. (ecBounceMessage)	0x000004D8
QuotaExceeded	The operation failed because it would have exceeded a resource quota. (ecQuotaExceeded)	0x000004D9
MaxSubmissionExceeded	A message could not be submitted because its size exceeds the defined maximum. (ecMaxSubmissionExceeded)	0x000004DA
MaxAttachmentExceeded	The maximum number of message attachments has been exceeded. (ecMaxAttachmentExceeded)	0x000004DB
SendAsDenied	The user account does not have permission to send mail as the owner of this mailbox. (ecSendAsDenied)	0x000004DC
ShutoffQuotaExceeded	The operation failed because it would have exceeded the mailbox's shutoff quota. (ecShutoffQuotaExceeded)	0x000004DD

TooManyOpenObjects	A client has opened too many objects of a specific type. (ecMaxObjsExceeded)	0x000004DE
ClientVersionBlocked	The server is configured to block clients of this version. (ecClientVerDisallowed)	0x000004DF
RpcHttpDisallowed	The server is configured to block RPC connections via HTTP. (ecRpcHttpDisallowed)	0x000004E0
CachedModeRequired	The server is configured to block online mode connections; only cached mode connections are allowed. (ecCachedModeRequired)	0x000004E1
FolderNotCleanedUp	The folder has been deleted but not yet cleaned up. (ecFolderNotCleanedUp)	0x000004E3
FormatError	Part of a ROP buffer was incorrectly formatted. (ecFmtError)	0x000004ED
NotExpanded	Error in expanding or collapsing rows in a categorized view. (ecNotExpanded)	0x000004F7
NotCollapsed	Error in expanding or collapsing rows in a categorized view. (ecNotCollapsed)	0x000004F8
NoExpandLeafRow	Leaf rows cannot be expanded; only category header rows can be expanded. (ecLeaf)	0x000004F9
UnregisteredNameProp	An operation was attempted on a named property ID for which no name has been registered. (ecUnregisteredNameProp)	0x000004FA
FolderDisabled	Access to the folder is disabled, perhaps because form design is in progress. (ecFolderDisabled)	0x000004FB

DomainError	There is an inconsistency in the message database's association with its server. (ecDomainError)	0x000004FC
NoCreateRight	The operation requires create access rights which the user does not have. (ecNoCreateRight)	0x000004FF
PublicRoot	The operation requires create access rights at a public folder root. (ecPublicRoot)	0x00000500
NoReadRight	The operation requires read access rights which the user does not have. (ecNoReadRight)	0x00000501
NoCreateSubfolderRight	The operation requires create subfolder access rights which the user does not have. (ecNoCreateSubfolderRight)	0x00000502
MessageCycle	The source message contains the destination message and cannot be attached to it. (ecMsgCycle)	0x00000504
NullDestinationObject	The RPC buffer contains a destination object handle that could not be resolved to a server object. (ecDstNullObject)	0x00000503
TooManyRecips	A hard limit on the number of recipients per message was exceeded. (ecTooManyRecips)	0x00000505
VirusScanInProgress	The operation failed because the target message is being scanned for viruses. (ecVirusScanInProgress)	0x0000050A
VirusDetected	The operation failed because the target message is infected with a virus. (ecVirusDetected)	0x0000050B
MailboxInTransit	The mailbox is in transit and is not accepting mail. (ecMailboxInTransit)	0x0000050C

BackupInProgress	The operation failed because the message database is being backed up. (ecBackupInProgress)	0x0000050D
VirusMessageDeleted	The operation failed because the target message was infected with a virus and has been deleted. (ecVirusMessageDeleted)	0x0000050E
InvalidBackupSequence	Backup steps were performed out of sequence. (ecInvalidBackupSequence)	0x0000050F
InvalidBackupType	The requested backup type was not recognized. (ecInvalidBackupType)	0x00000510
TooManyBackups	Too many backups are already in progress. (ecTooManyBackupsInProgress)	0x00000511
RestoreInProgress	A restore is already in progress. (ecRestoreInProgress)	0x00000512
DuplicateObject	The object already exists. (ecDuplicateObject)	0x00000579
ObjectNotFound	An internal database object could not be found. (ecObjectNotFound)	0x0000057A
FixupReplyRule	A rule object MUST be fixed up with the reply template message ID. (ecFixupReplyRule)	0x0000057B
TemplateNotFound	The reply template could not be found for a message that triggered an auto-reply rule. (ecTemplateNotFound)	0x0000057C
RuleExecution	An error occurred while executing a rule action. (ecRuleExecution)	0x0000057D
DSNoSuchObject	A server object could not be found in the directory. (ecDSNoSuchObject)	0x0000057E

AlreadyTombstoned	An attempt to tombstone a message already in the message tombstone list failed. (ecMessageAlreadyTombstoned)	0x0000057F
ReadOnlyTransaction	A write operation was attempted in a read-only transaction. (ecRequiresRWTransaction)	0x00000596
Paused	Attempt to pause a server that is already paused. (ecPaused)	0x0000060E
NotPaused	Attempt to unpause a server that is not paused. (ecNotPaused)	0x0000060F
WrongMailbox	The operation was attempted on the wrong mailbox. (ecWrongMailbox)	0x00000648
ChangePassword	The account password MUST be changed. (ecChgPassword)	0x0000064C
PasswordExpired	The account password has expired. (ecPwdExpired)	0x0000064D
InvalidWorkstation	The account has logged on from the wrong workstation. (ecInvWkstn)	0x0000064E
InvalidLogonHours	The account has logged on at the wrong time of day. (ecInvLogonHrs)	0x0000064F
AccountDisabled	The account is disabled. (ecAcctDisabled)	0x00000650
RuleVersion	The rule data contains an invalid rule version. (ecRuleVersion)	0x000006A4
RuleFormat	The rule condition or action was incorrectly formatted. (ecRuleFormat)	0x000006A5

RuleSendAsDenied	The rule is not authorized to send from this mailbox. (ecRuleSendAsDenied)	0x000006A6
NoServerSupport	A newer client requires functionality that an older server does not support. (ecNoServerSupport)	0x000006B9
LockTimedOut	An attempt to unlock a message failed because the lock had already timed out. (ecLockTimedOut)	0x000006BA
ObjectLocked	The operation failed because the target object is locked. (ecObjectLocked)	0x000006BB
InvalidLockNamespace	Attempt to lock a nonexistent object. (ecInvalidLockNamespace)	0x000006BD
MessageDeleted	Operation failed because the message has been deleted. (ecMessageDeleted)	0x000007D6
ProtocolDisabled	The requested protocol is disabled in the server configuration. (ecProtocolDisabled)	0x000007D8
CleartextLogonDisabled	Clear text logons were disabled. (ecCleartextLogonDisabled)	0x000007D9
Rejected	The operation was rejected, perhaps because it is not supported. (ecRejected)	0x000007EE
AmbiguousAlias	User account information did not uniquely identify a user. (ecAmbiguousAlias)	0x0000089A
UnknownMailbox	No mailbox object for this logon exists in the address book. (ecUnknownMailbox)	0x0000089B
ExpressionReserved	Internal error in evaluating an expression. (ecExpReserved)	0x000008FC

ExpressionParseDepth	The expression tree exceeds a defined depth limit. (ecExpParseDepth)	0x000008FD
ExpressionArgumentType	An argument to a function has the wrong type. (ecExpFuncArgType)	0x000008FE
ExpressionSyntax	Syntax error in expression. (ecExpSyntax)	0x000008FF
ExpressionBadStringToken	Invalid string token in expression. (ecExpBadStrToken)	0x00000900
ExpressionBadColToken	Invalid column name in expression. (ecExpBadColToken)	0x00000901
ExpressionTypeMismatch	Property types in, for example, a comparison expression, are incompatible. (ecExpTypeMismatch)	0x00000902
ExpressionOperatorNotSupported	The requested operator is not supported. (ecExpOpNotSupported)	0x00000903
ExpressionDivideByZero	Divide by zero doesn't work. (ecExpDivByZero)	0x00000904
ExpressionUnaryArgument	The argument to a unary expression is of incorrect type. (ecExpUnaryArgType)	0x00000905
NotLocked	An attempt to lock a resource failed. (ecNotLocked)	0x00000960
ClientEvent	A client-suplied event has fired. (ecClientEvent)	0x00000961
CorruptEvent	Data in the event table is bad. (ecCorruptEvent)	0x00000965
CorruptWatermark	A watermark in the event table is bad. (ecCorruptWatermark)	0x00000966
EventError	General event processing error. (ecEventError)	0x00000967

WatermarkError	An event watermark is out of range or otherwise invalid. (ecWatermarkError)	0x00000968
NonCanonicalACL	A modification to an access control list failed because the existing ACL is not in canonical format. (ecNonCanonicalACL)	0x00000969
MailboxDisabled	Logon was unsuccessful because the mailbox is disabled. (ecMailboxDisabled)	0x0000096C
RulesFolderOverQuota	A move or copy rule action failed because the destination folder is over quota. (ecRulesFolderOverQuota)	0x0000096D
AddressBookUnavailable	The address book server could not be reached. (ecADUnavailable)	0x0000096E
AddressBookError	Unspecified error from the Address Book server. (ecADError)	0x0000096F
AddressBookObjectNotFound	An object was not found in the Address Book. (ecADNotFound)	0x00000971
AddressBookPropertyError	A property was not found in the Address Book. (ecADPropertyError)	0x00000972
NotEncrypted	The server is configured to force encrypted connections, but the client requested an unencrypted connection. (ecNotEncrypted)	0x00000970
RpcServerTooBusy	An external RPC call failed because the server was too busy. (ecRpcServerTooBusy)	0x00000973

RpcOutOfMemory	An external RPC call failed because the local server was out of memory. (ecRpcOutOfMemory)	0x00000974
RpcServerOutOfMemory	An external RPC call failed because the remote server was out of memory. (ecRpcServerOutOfMemory)	0x00000975
RpcOutOfResources	An external RPC call failed because the remote server was out of an unspecified resource. (ecRpcOutOfResources)	0x00000976
RpcServerUnavailable	An external RPC call failed because the remote server was unavailable. (ecRpcServerUnavailable)	0x00000977
SecureSubmitError	A failure occurred while setting the secure submission state of a message. (ecSecureSubmitError)	0x0000097A
EventsDeleted	Requested events were already deleted from the queue. (ecEventsDeleted)	0x0000097C
SubsystemStopping	A component service is in the process of shutting down. (ecSubsystemStopping)	0x0000097D
AttendantUnavailable	The system attendant service is unavailable. (ecSAUnavailable)	0x0000097E
CIStopping	The content indexer service is stopping. (ecCIStopping)	0x00000A28
FxInvalidState	An internal fast transfer object has invalid state. (ecFxInvalidState)	0x00000A29
FxUnexpectedMarker	Fast Transfer parsing has hit an invalid marker. (ecFxUnexpectedMarker)	0x00000A2A

DuplicateDelivery	A copy of this message has already been delivered. (ecDuplicateDelivery)	0x00000A2B
ConditionViolation	The condition was not met for a conditional operation. (ecConditionViolation)	0x00000A2C

2.4.2 Property Error Codes

Property errors appear in two different contexts. When an error occurs in getting a property of an object, or a column of a table, from the server, then the type of the returned property value is ErrorCode (0x000A) and the property value itself is the error code. When an error occurs in setting a property of an object on the server, then the **RopSetProperties** returns an array of PropertyProblem structures that includes the error code.

Most property error codes are also used as general error codes, but they have a special meaning in the context of a property operation.

Property Error Codes are presented in the following table:

Table 7 Property Error Codes

Name	Description (alternate names)	Numeric value (hex)
NotEnoughMemory	On get, indicates that the property or column value is too large to be retrieved by the request, and property value MUST instead be accessed with a stream interface. (E_NOMEMORY, MAPI_E_NOT_ENOUGH_MEMORY)	8007000E, %x0E.00.07.80
NotFound	On get, indicates that the property or column has no value for this object. (MAPI_E_NOT_FOUND)	8004010F, %x0F.01.04.80
BadValue	On set, indicates that the property value is not acceptable to the server. (MAPI_E_BAD_VALUE, ecPropBadValue)	80040301, %x01.03.04.80

InvalidType	On get or set, indicates that the data type passed with the property or column is undefined. (MAPI_E_INVALID_TYPE, ecInvalidType)	80040302, %x02.03.04.80
UnsupportedType	On get or set, indicates that the data type passed with the property or column is not acceptable to the server. (MAPI_E_TYPE_NO_SUPPORT, ecTypeNotSupported)	80040303, %x03.0.04.80
UnexpectedType	On get or set, indicates that the data type passed with the property or column is not the type expected by the server. (MAPI_E_UNEXPECTED_TYPE, ecPropType)	80040304, %x04.03.04.80
TooBig	Indicates that the result set of the operation is too big for the server to return. (MAPI_E_TOO_BIG, ecTooBig)	80040305, %x05.03.04.80
DeclineCopy	On a copy operation, indicates that the server cannot copy the object – possibly because the source and destination are on different types of servers – and wishes to delegate the copying to client code. (MAPI_E_DECLINE_COPY)	80040306, %x06.03.04.80
UnexpectedId	On get or set, indicates that the server does not support property IDs in this range, usually the named property ID range (0x8000-0xFFFF). (MAPI_E_UNEXPECTED_ID)	80040307, %x07.03.04.80

2.4.3 Warning Codes

Warning codes indicate that while the operation as a whole was processed successfully by the server, individual items or properties were not processed successfully. For example, if three properties are requested from a message object in a **RopGetPropertiesSpecific** operation and one of the three properties does not exist on the message object, then in the return buffer:

a) The ROP returns an ErrorsReturned warning.

b) The type in the property tag of the missing property is errorcode.

c) The property value of the missing property is notfound.

Warning codes are presented in the following table:

Table 8 Warning Codes

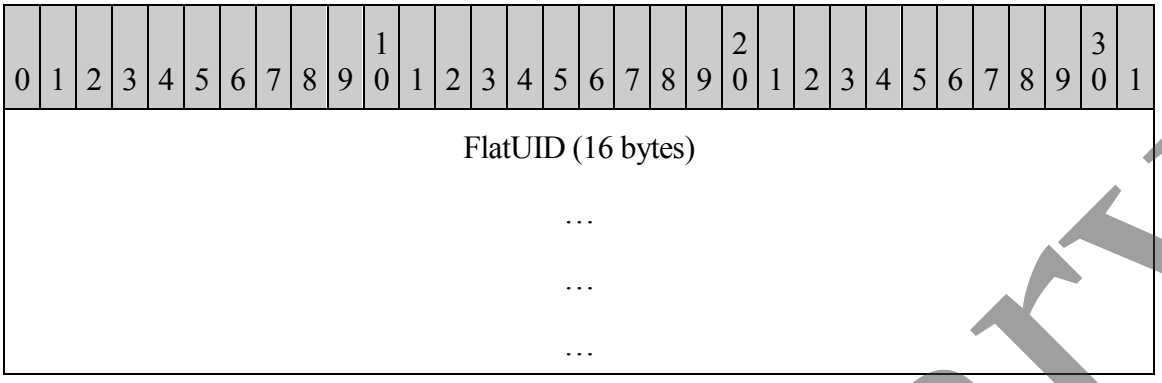
Name	Description (alternate names)	Numeric value (hex)
ErrorsReturned	A request involving multiple properties failed for one or more individual properties, while succeeding overall. (MAPI_W_ERRORS_RETURNED, ecWarnWithErrors)	00040380, %x80.03.04.00
PositionChanged	A table operation succeeded, but the bookmark specified is no longer set at the same row as when it was last used. (MAPI_W_POSITION_CHANGED, ecWarnPositionChanged)	00040481, %x81.04.04.00
ApproximateCount	The row count returned by a table operation is approximate, not exact. (MAPI_W_APPROX_COUNT, ecWarnApproxCount)	00040482, %x82.04.04.00
PartiallyComplete	A move, copy, or delete operation succeeded for some messages but not for others. (MAPI_W_PARTIAL_COMPLETION, ecPartialCompletion)	00040680, %x80.06.04.00
SyncProgress	The operation succeeded but there is more to do. (SYNC_W_PROGRESS)	00040820, %x20.08.04.00
NewerClientChange	In a change conflict, the client has the more recent change. (SYNC_W_CLIENT_CHANGE_NEWER)	00040821, %x21.08.04.00

2.5 Flat UID

The FlatUID structure is a byte-order independent version of a GUID structure and is used to uniquely identify a service provider. It appears in EntryIds.

A FlatUID is a GUID structure put into little-endian byte order. That is, FlatUID and GUID structures have the same byte order when used on a little-endian processor. However, on a

big-endian processor, the FlatUID has the same byte order as on the little-endian machine, but the GUID uses big-endian format for certain fields



FlatUID (16 bytes): A flat 16-byte little-endian sequence used as a unique identifier in various structures.

2.6 Notifications

A client can use RopRegisterNotification to request a server to notify it when one of the following events occurs:

- New mail delivery
- Object creation
- Object modification
- Object deletion
- Search completion
- Contents or hierarchy table change

In addition a client can use **RopRegisterSynchronizationNotifications** to request a server to notify it when the following event occurs:

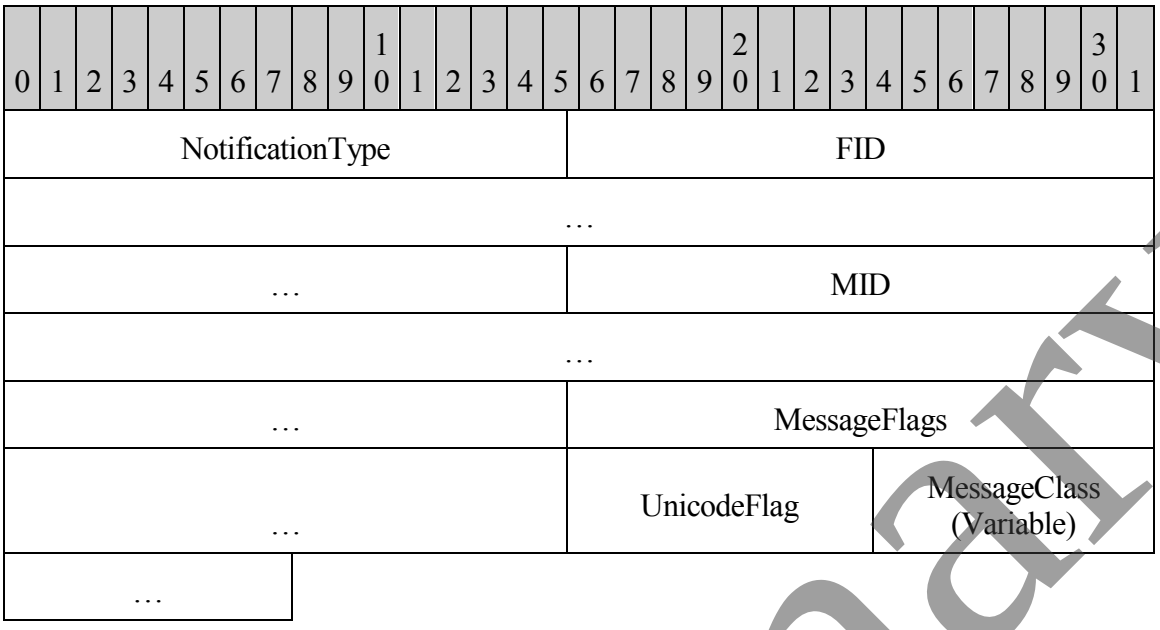
- Folder(s) need to be resynched

The server queues notifications for the client as events occur, and when the client makes an RPC call (of any sort), the pending notifications are returned as individual **RopNotify** packets as part of the response to the RPC call.

This section describes the formats of the notification structures returned by servers.

2.6.1 New Mail Delivery

A client MAY request notification of new mail delivery to a mailbox store or to a specified folder within a mailbox store or the public store. When a new message is delivered, the server creates a **NewMailNotification** structure for the client.



NotificationType (2 bytes): Unsigned 16-bit integer. This value indicates the type of notification and in this case MUST be set to 0x2 by the server.

FID (8 bytes): FID structure**.** The value identifies the folder where the new mail was received.

MID (8 bytes): MID structure**.** The value identifies the new mail message.

MessageFlags (4 bytes): Unsigned 4-byte integer giving the value of the PidTagMessageFlags property for the new mail message.

UnicodeFlag (1 byte): Unsigned 1-byte integer. If TRUE (0x01), indicates that the MessageClass field which follows is a Unicode (UTF-16) string; otherwise, the MessageClass field is an MBCS string.

MessageClass (Variable): A null-terminated string which is in Unicode if the value of **UnicodeFlag** is TRUE (0x01) or in MBCS if the value of **UnicodeFlag** is FALSE (0x00).

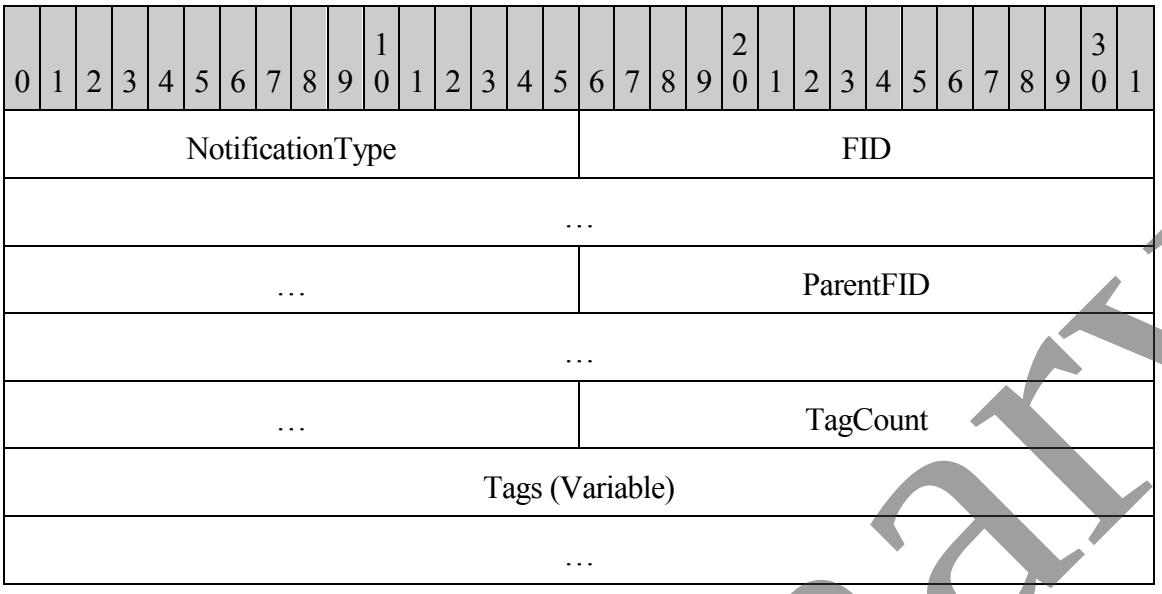
2.6.2 Object Creation

A client MAY request notification when objects are created within an entire mailbox store or as immediate child objects of a specified folder. Note: Move and Copy operations generate object creation notifications for the destination folder.

The next subsections describe the three different formats for object creation notifications.

2.6.2.1 FolderCreatedNotification

This structure is used to notify a client that a new folder has been created.



NotificationType (2 bytes): Unsigned 16-bit integer. This value indicates the type of notification and MUST be set to 0x0004 by the server.

FID (8 bytes): FID structure**.** The value identifies the new folder which was created.

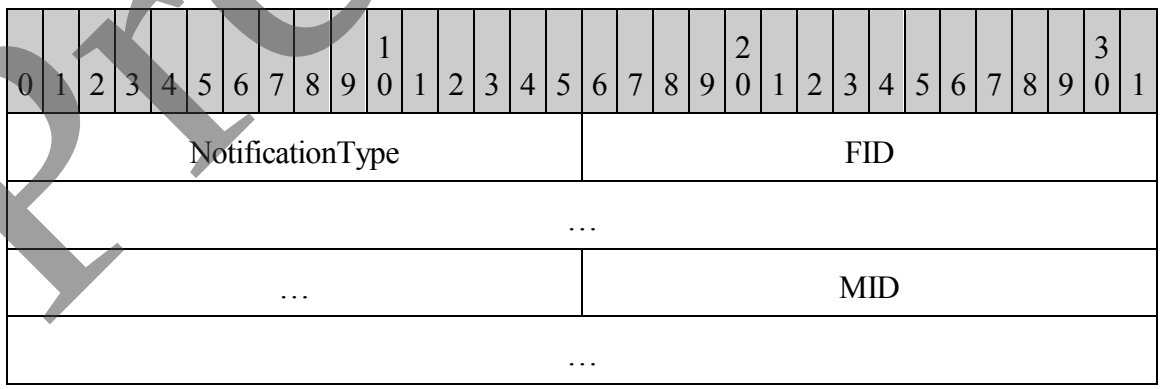
ParentFID (8 bytes): FID structure**.** The value identifies the parent folder which contains the new folder.

TagCount (2 bytes): Unsigned 16-bit integer. When it has the value 0xFFFF, the Tags field is not present. Otherwise, it indicates how many PropertyTag elements are present in Tags.

Tags (variable): Array of PropertyTag structures. This field MUST contain TagCount tags; it simply lists properties that were set on the new folder, without specifying their values.

2.6.2.2 MessageCreatedNotification

This structure is used to notify a client when a new message is created in a normal folder. There is a different notification type for search folders.



…	TagCount
Tags (Variable)	
…	

NotificationType (2 bytes): Unsigned 16-bit integer. This value indicates the type of notification and MUST be set to 0x8004 by the server.

FID (8 bytes): FID structure**.** The value identifies the folder containing the message.

MID (8 bytes): MID structure**.** The value identifies the message.

TagCount (2 bytes): Unsigned 16-bit integer. When it has the value 0xFFFF, the Tags field is not present. Otherwise, it indicates how many PropertyTag elements are present in Tags.

Tags (variable): Array of PropretyTag structures. This field MUST contain TagCount structures. It lists properties that were initially set or updated on the message.

2.6.2.3 SearchMessageAddedNotification

This structure is used to notify when a new message is added to the search results in a search folder. When the server discovers a message that meets the folder's search criteria, it inserts a link to the message into the search folder's contents table and generates this notification for the client.

0 1 2 3 4 5 6 7 8 9 10 1 2 3 4 5 6 7 8 9 20 1 2 3 4 5 6 7 8 9 30 1	
NotificationType	FID
…	
…	MID
…	
…	SearchFID
…	
…	TagCount
Tags (Variable)	
…	

NotificationType (2 bytes): Unsigned 16-bit integer. This value indicates the type of notification and MUST be set to 0xC004 by the server.

FID (8 bytes): FID structure**.** The value identifies the folder the message is actually stored in, and not the search folder itself.

MID (8 bytes): MID structure**.** The value identifies the message.

SearchFID (8 bytes): FID structure**.** The value identifies the search folder the message was added to.

TagCount (2 bytes): Unsigned 16-bit integer. When it has the value 0xFFFF, the Tags field is not present. Otherwise, it indicates how many PropertyTag elements are present in Tags.

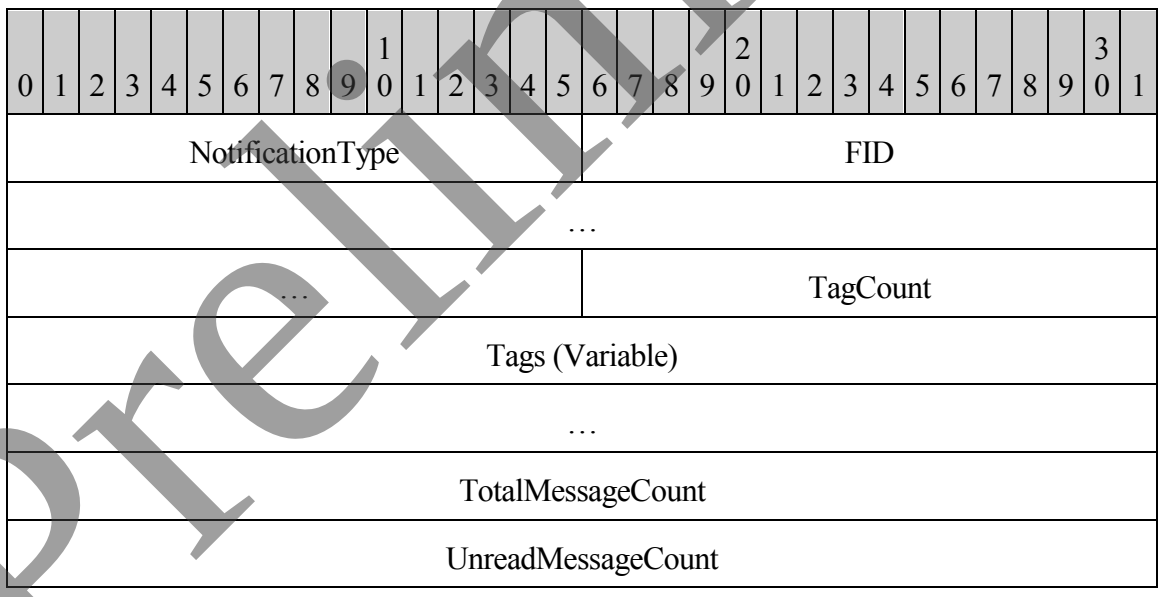
Tags (variable): Array of PropertyTag structures. This field MUST contain TagCount tags. It lists properties that were initially set or updated on the message.

2.6.3 Object Modification

A client MAY request notification about the changes to properties of objects. The following subsections describe the three different formats for object modification notifications.

2.6.3.1 FolderModifiedNotification

This structure is used to notify a client when the properties of a folder change.



NotificationType (2 bytes): Unsigned 16-bit integer. This value indicates the type of notification and MUST be set to one of four values: 0x0010, 0x1010, 0x2010, or 0x3010 by

the server. These different values indicate the presence or absence of the TotalMessageCount and UnreadMessageCount fields.

FID (8 bytes): FID structure**.** The value identifies the folder which was modified.

TagCount (2 bytes): Unsigned 16-bit integer. When it has the value 0xFFFF, the Tags field is not present. Otherwise, it indicates how many PropertyTag elements are present in Tags.

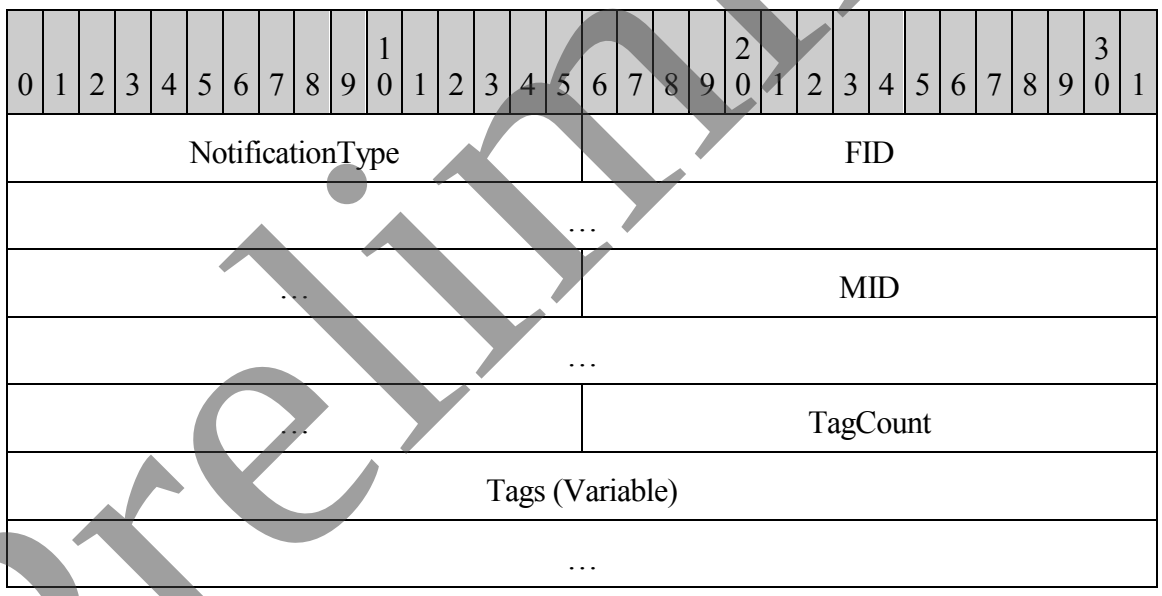
Tags (variable): Array of PropertyTag structures. This field MUST contain TagCount structures. This field gives a list of properties that were changed on the folder.

TotalMessageCount (4 bytes): Unsigned 32-bit integer. This field is ONLY present when NotificationType is 0x1010 or 0x3010. This value gives the new number of messages in the folder.

UnreadMessageCount (4 bytes): Unsigned 32-bit integer. This field is ONLY present when NotificationType is 0x2010 or 0x3010. This value gives the new number of unread messages in the folder.

2.6.3.2 MessageModifiedNotification

This structure is used to notify a client when the properties of a message in a normal, non-search folder change. A different type of notification is issued for search folders.



NotificationType (2 bytes): Unsigned 16-bit integer. This value indicates the type of notification and MUST be set to 0x8010 by the server.

FID (8 bytes): FID structure**.** The value identifies the folder containing the message.

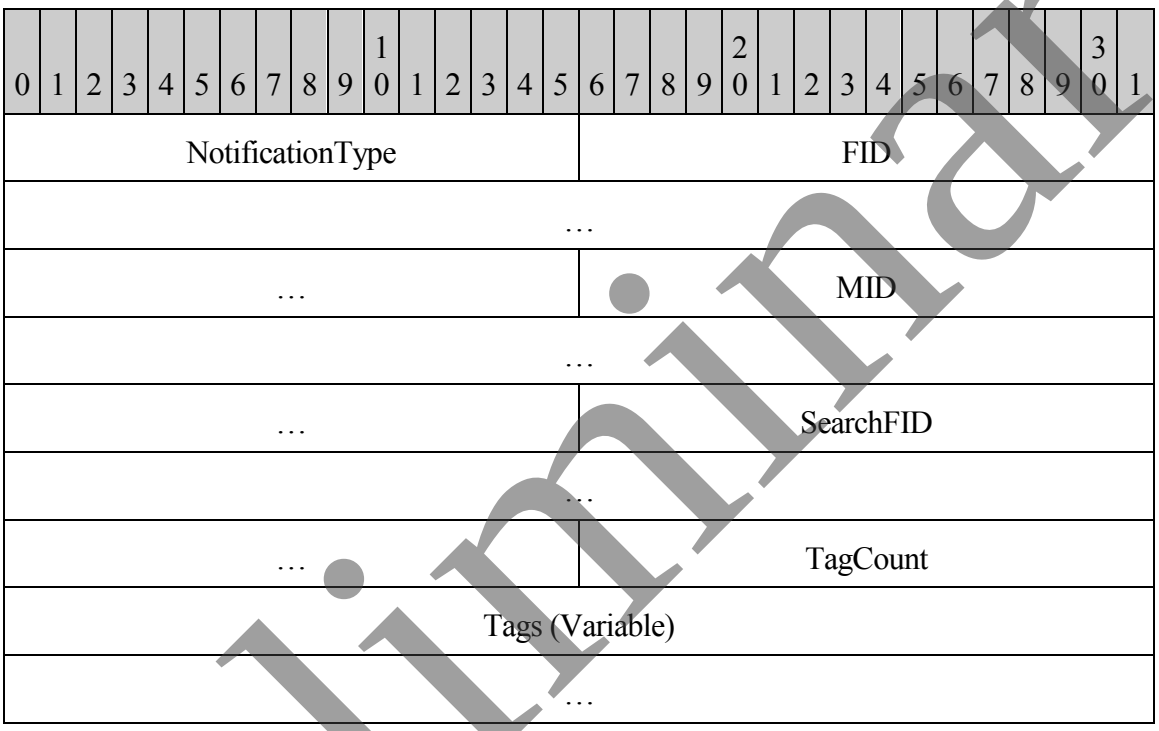
MID (8 bytes): MID structure**.** The value identifies the message.

TagCount (2 bytes): Unsigned 16-bit integer. When it has the value 0xFFFF, the Tags field is not present. Otherwise, it indicates how many PropertyTag elements are present in Tags.

Tags (variable): Array of PropertyTag structures. This field MUST contain TagCount structures. This field gives a list of properties that were initially set or updated on the message.

2.6.3.3 SearchMessageModifiedNotification

This structure is used to notify a client when the properties of a message in a search folder change (but the message still meets the folder's search criteria).



NotificationType (2 bytes): Unsigned 16-bit integer. This value indicates the type of notification and MUST be set to 0xC010 by the server.

FID (8 bytes): FID structure**.** The value identifies the folder the message is actually stored in.

MID (8 bytes): MID structure**.** The value identifies the message.

SearchFID (8 bytes): FID structure**.** The value identifies the search folder the message is in.

TagCount (2 bytes): Unsigned 16-bit integer. When it has the value 0xFFFF, the Tags field is not present. Otherwise, it indicates how many PropertyTag elements are present in Tags.

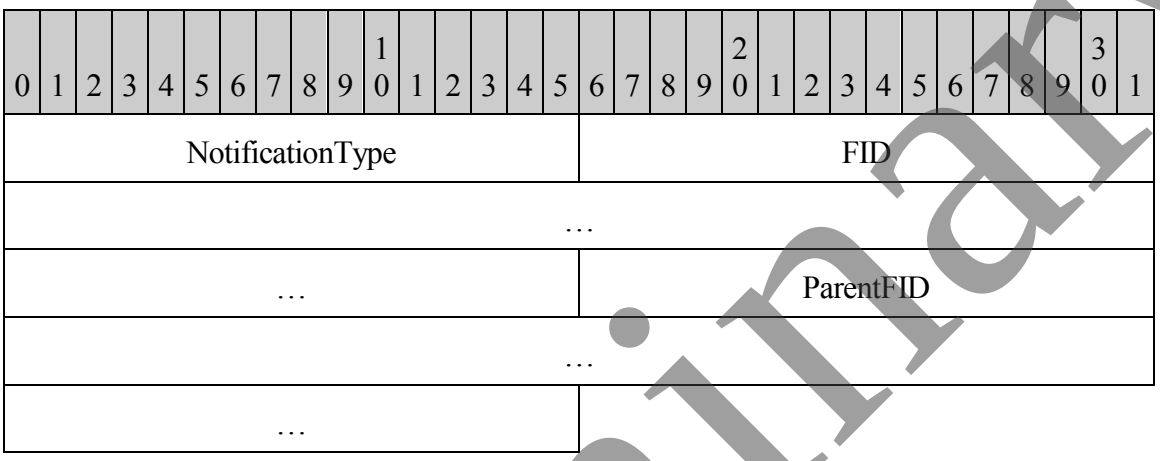
Tags (variable): Array of PropertyTag structures. This field MUST contain PidTagCount structures. This field gives a list of properties that were initially set or updated on the message.

2.6.4 Object Deletion

A client MAY request notification about deletion of objects. The following subsections describe the three different formats for object deletion notifications. Note: Move operations generate object deletion notifications for the source folder.

2.6.4.1 FolderDeletedNotification

This structure is used to notify a client when a folder has been deleted.



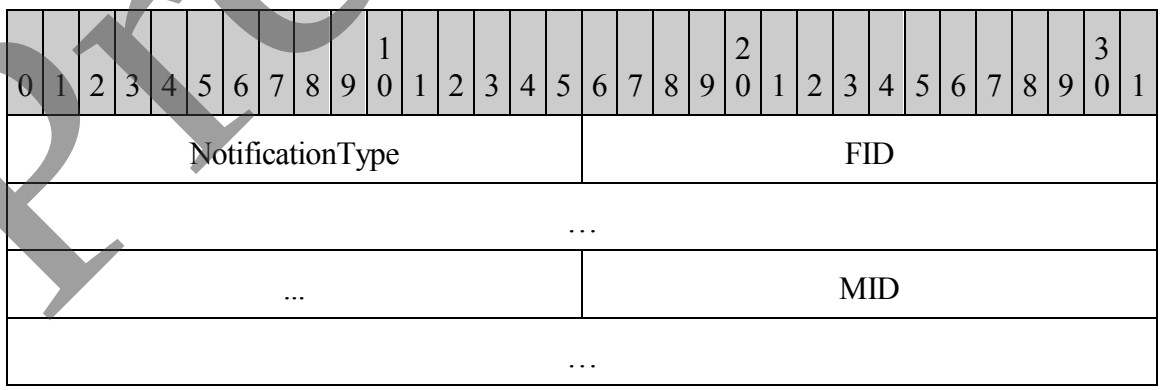
NotificationType (2 bytes): Unsigned 16-bit integer. This value indicates the type of notification and in this case MUST be set to 0x0008 by the server.

FID (8 bytes): FID structure**.** The value identifies the folder which was deleted.

ParentFID (8 bytes): FID structure**.** The value identifies the parent folder which used to contain the deleted folder.

2.6.4.2 MessageDeletedNotification

This structure is used to notify a client when a message has been deleted.



…	

NotificationType (2 bytes): Unsigned 16-bit integer. This value indicates the type of notification and in this case MUST be set to 0x8008 by the server.

FID (8 bytes): FID structure**.** The value identifies the folder containing the message which was deleted.

MID (8 bytes): MID structure**.** The value identifies the message which was deleted.

2.6.4.3 SearchMessageRemovedNotification

This structure is used to notify clients that a message is no longer part of a search folder. This can occur because the message was actually deleted, or because properties of the message have changed in such a way that it no longer meets the search criteria.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
NotificationType																FID															
…																															
...																MID															
…																															
…																SearchFID															
…																															
…																															

NotificationType (2 bytes): Unsigned 16-bit integer. This value indicates the type of notification and in this case MUST be set to 0xC008 by the server.

FID (8 bytes): FID structure**.** The value identifies the folder that contained the message (if it was deleted) or still contains the message (if its properties have changed in such a way that it no longer meets the search criteria).

MID (8 bytes): MID structure**.** The value identifies the message.

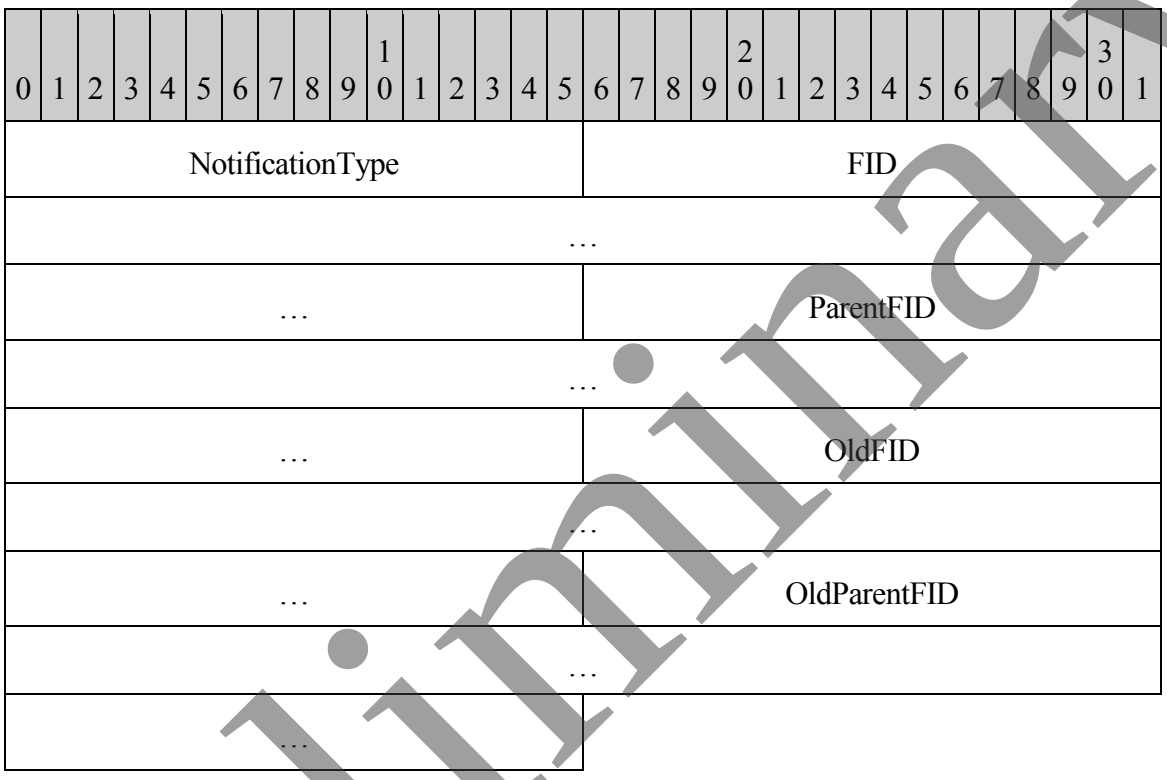
SearchFID (8 bytes): FID structure**.** The value identifies the search folder the message has been removed from.

2.6.5 Object Moved or Copied

A client MAY request notification about objects that are moved or copied. The following subsections describe the two different formats for object modification notifications.

2.6.5.1 FolderMoveCopyNotification

This structure is used to notify clients when a folder is moved or copied.



NotificationType (2 bytes): Unsigned 16-bit integer. This value indicates the type of notification and MUST be set by the server to 0x0020 to indicate a move notification, or 0x0040 to indicate a copy notification.

FID (8 bytes): FID structure**.** The value gives the new FID of the folder which was moved or copied.

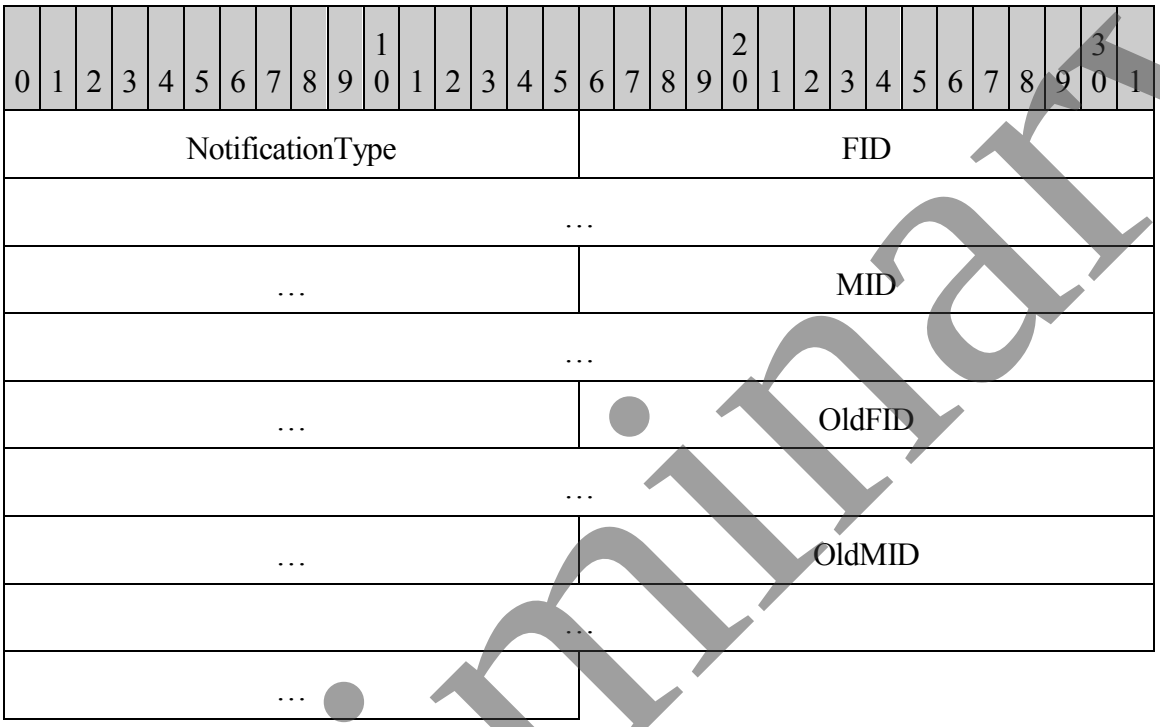
ParentFID (8 bytes): FID structure**.** The value identifies the parent folder which now contains the folder which was moved or copied.

OldFID (8 bytes): FID structure**.** In the case of a move notification, this gives the FID of the folder before it was moved. In the case of copy notification, this gives the FID of the folder which was copied.

OldParentFID (8 bytes): FID structure**.** The value identifies the parent folder from which the folder was moved or copied.

2.6.5.2 MessageMoveCopyNotification

This structure is used to notify clients when a message is moved or copied.



NotificationType (2 bytes): Unsigned 16-bit integer. This value indicates the type of notification and MUST be set by the server to 0x8020 to indicate a move notification, or 0x8040 to indicate a copy notification.

FID (8 bytes): FID structure**.** The value identifies the new parent folder of the message which was moved or copied.

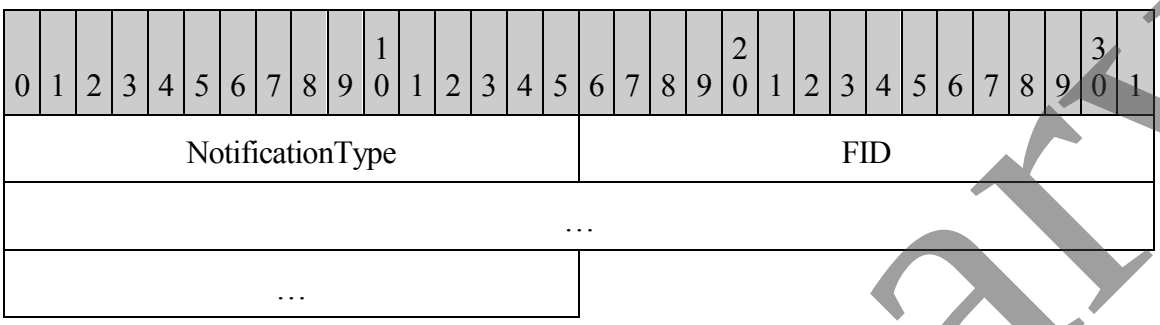
MID (8 bytes): MID structure**.** The value gives the new MID of the message that was moved or copied.

OldFID (8 bytes): FID structure**.** The value identified the parent folder that the message was moved or copied from.

OldMID (8 bytes): MID structure**.** In the case of a move notification, this gives the MID of the message before it was moved. In the case of copy notification, this gives the MID of the message which was copied.

2.6.6 Search Complete

A client MAY request notification when a search is complete, that is, when all messages within the scope of the search have been evaluated. When this occurs, a SearchCompleteNotification structure is used to notify the client.



NotificationType (2 bytes): Unsigned 16-bit integer. This value indicates the type of notification and MUST be set to 0x0080 by the server.

FID (8 bytes): FID structure**.** The value identifies the search folder where the search has completed.

2.6.7 Contents or Hierarchy Table Change

A client can register for notification on changes to an open hierarchy or contents table object. Servers SHOULD generate rich table notifications that allow the client to update its view of the table without making additional queries for row information. The following table lists table notification types:

Table 9 Rich Table Notification Types

Name	Value	Description
TABLE_ROW_ADDED	3	A new row has been added to the table.
TABLE_ROW_DELETED	4	A row has been removed from the table
TABLE_ROW_MODIFIED	5	One or more property values in a row have been changed.
TABLE_CHANGED	1	When it is not possible to generate a rich notification, the server SHOULD generate this notification. Indicates at a high level that something about the table has changed. The table's state is as it was before the event, meaning that the values of the instance key

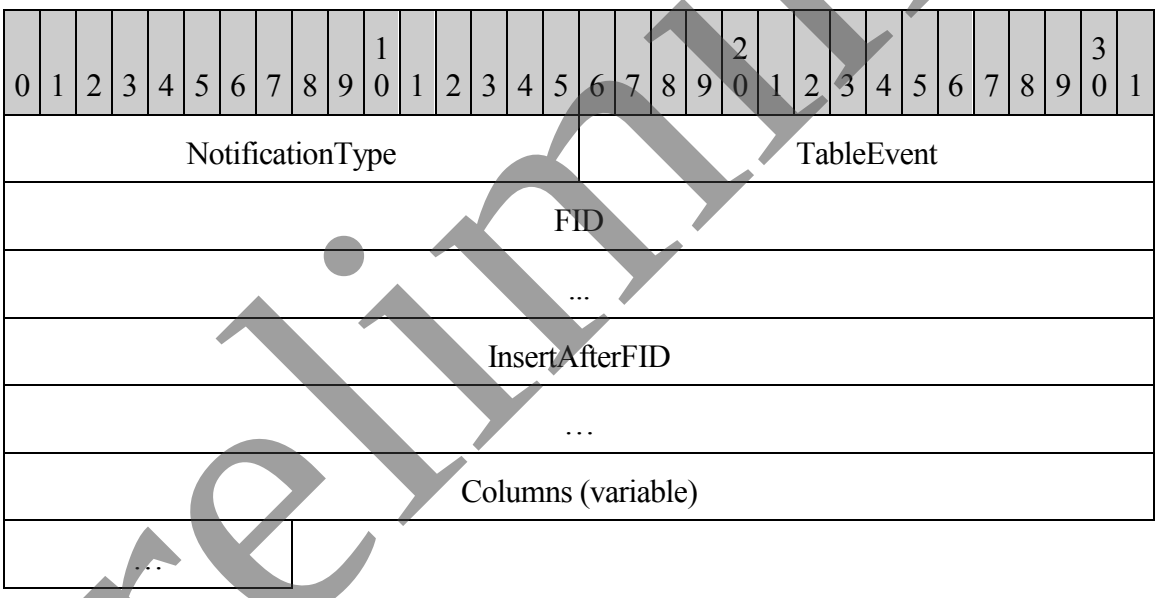
		property, bookmarks, current positioning, and user interface selections are still valid, but the table MUST be reread from the server.
TABLE_RESTRICT_DONE	7	Servers MUST generate the this table event after the computation of a new view has been completed: Indicates that an asynchronous restriction operation has completed.

2.6.7.1 TABLE_ROW_ADDED Notifications

A client can use a TABLE_ROW_ADDED notification to directly update its row cache without pulling new data from the server.

There are two formats for TABLE_ROW_ADDED notifications, one for hierarchy tables and another for contents tables.

2.6.7.1.1 HierarchyRowAddedNotification



NotificationType (2 bytes): Unsigned 16-bit integer. This value indicates the type of notification and MUST be set to 0x0100 by the server.

TableEvent (2 bytes): Unsigned 16-bit integer. This value indicates the type of table event and MUST be set to 0x0003 by the server.

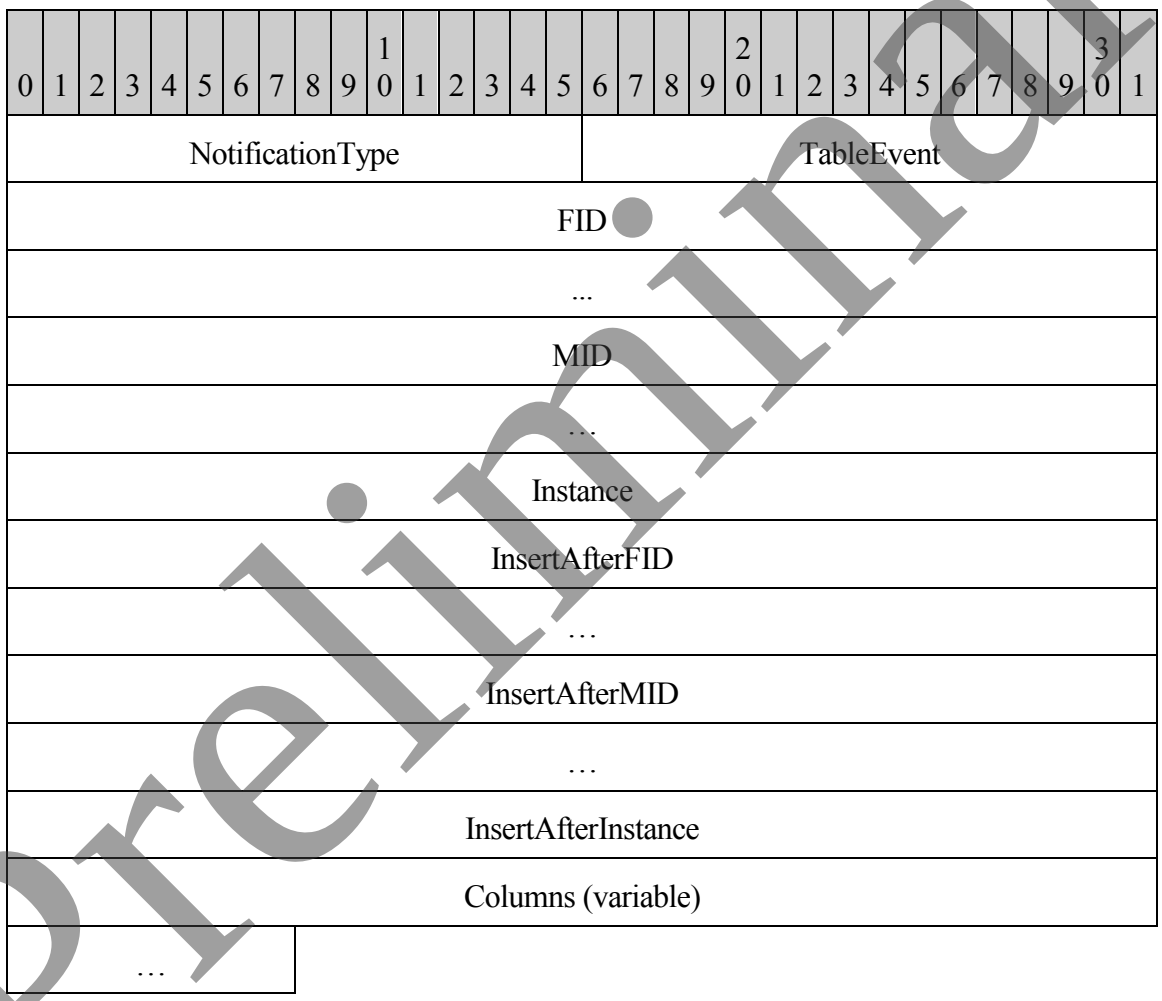
FID (8 bytes): FID structure. This value gives the FID of the folder that was added.

InsertAfterFID (8 bytes): FID structure. This value gives the FID of the folder row that the new row SHOULD be inserted after. If the new folder row SHOULD be placed first, then this field MUST consist of 8 zero bytes.

Columns (variable): PropertyRow structure, as specified in section 2.10. This field gives the values for the columns of the new row.

The FID value is the unique identifier for a row of a hierarchy table. In order to update its view of a hierarchy table when a new row notification arrives, the client SHOULD cache the rows it is currently displaying, and store the FID with each row.

2.6.7.1.2 ContentsRowAddedNotification



NotificationType (2 bytes): Unsigned 16-bit integer. This value indicates the type of notification and MUST be set to 0x8100 by the server.

TableEvent (2 bytes): Unsigned 16-bit integer. This value indicates the type of table event and MUST be set to 0x003 by the server.

FID (8 bytes): FID structure. This value gives the FID of the row that was added.

MID (8 bytes): MID structure. When the table is categorized, this gives the category id of the new row. In other cases it gives the MID of the new row.

Instance (4 bytes): Unsigned 32-bit integer. When the table is categorized and one of the columns is multi-valued and specified as MultivalueInstance, this gives the instance number of the new row. In other cases it will be 0.

InsertAfterFID (8 bytes): FID structure. This value gives the FID of the row that the new row SHOULD be inserted after. If the new folder row SHOULD be placed first, then this field will consist of 8 zero bytes.

InsertAfterMID (8 bytes): MID structure. This value gives the MID (or category id) of the row that the new row SHOULD be inserted after. If the new folder row SHOULD be placed first, then this field will consist of 8 zero bytes.

InsertAfterInstance (4 bytes): Unsigned 32-bit integer. This value gives the instance value for the row that new row SHOULD be inserted after. If the new folder row SHOULD be placed first, then this value will be zero.

Columns (variable): PropertyRowSet structure. This field gives the values for the columns of the new row.

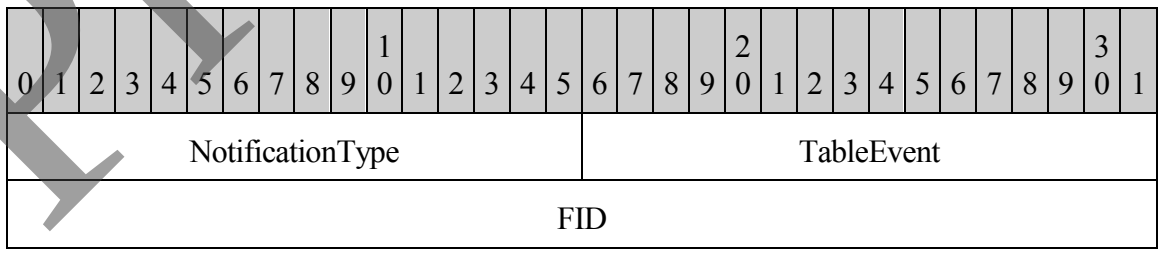
The combination of the FID**,** MID, and Instance makes a unique identifier for a row of a contents table. In order to update its view of a contents table when a new row notification arrives, a client SHOULD cache the rows it is currently displaying, and store the FID, MID, and Instance values with each row.

2.6.7.2 TABLE_ROW_DELETED Notifications

A client can use a TABLE_ROW_DELETED notification to directly update its row cache without pulling new data from the server.

There are two formats for TABLE_ROW_DELETED notifications: one for hierarchy tables and another for contents tables.

2.6.7.2.1 HierarchyRowDeletedNotification



...

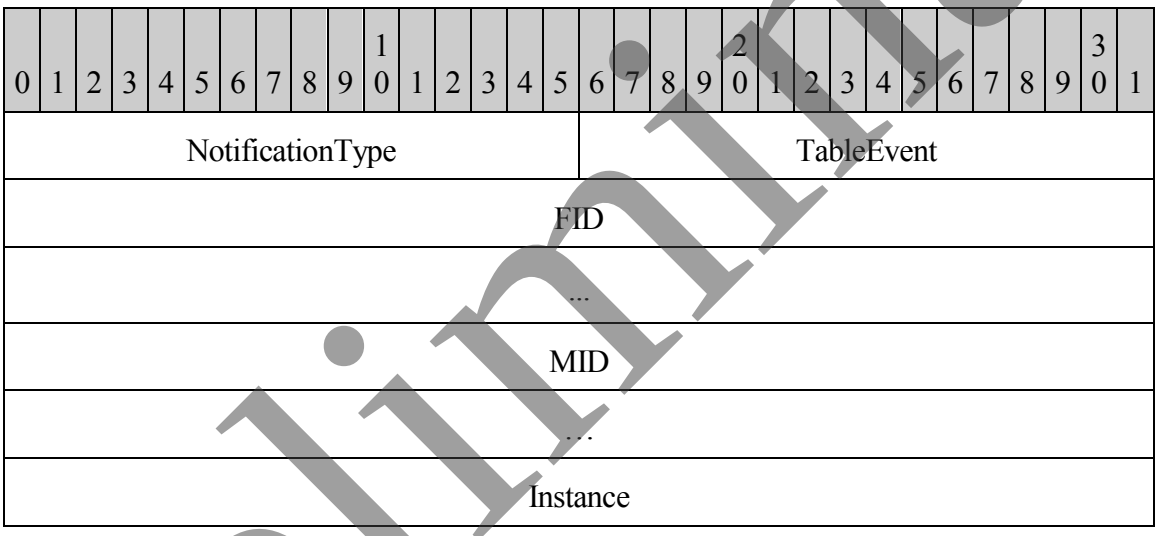
NotificationType (2 bytes): Unsigned 16-bit integer. This value indicates the type of notification and MUST be set to 0x0100 by the server.

TableEvent (2 bytes): Unsigned 16-bit integer. This value indicates the type of table event and MUST be set to 0x0004 by the server.

FID (8 bytes): FID structure. This value gives the FID of the folder row that was deleted.

The FID value is the unique identifier for a row of a hierarchy table. In order to update its view of a hierarchy table when a new row notification arrives, the client SHOULD cache the rows it is currently displaying, and store the FID with each row.

2.6.7.2.2 ContentsRowDeletedNotification



NotificationType (2 bytes): Unsigned 16-bit integer. This value indicates the type of notification and MUST be set to 0x8100 by the server.

TableEvent (2 bytes): Unsigned 16-bit integer. This value indicates the type of table event and MUST be set to 0x0004 by the server.

FID (8 bytes): FID structure. This value gives the FID of the row that was deleted.

MID (8 bytes): MID structure. When the table is categorized, this gives the category id of the row that was deleted. In other cases, it gives the MID of the deleted row.

Instance (4 bytes): Unsigned 32-bit integer. When the table is categorized and one of the columns is multi-valued and specified as MULTIVALUEINSTANCE, this gives the instance

number of the deleted row; the instance number is the value of PidTagInstanceKey column in that row. In other cases it will be 0.

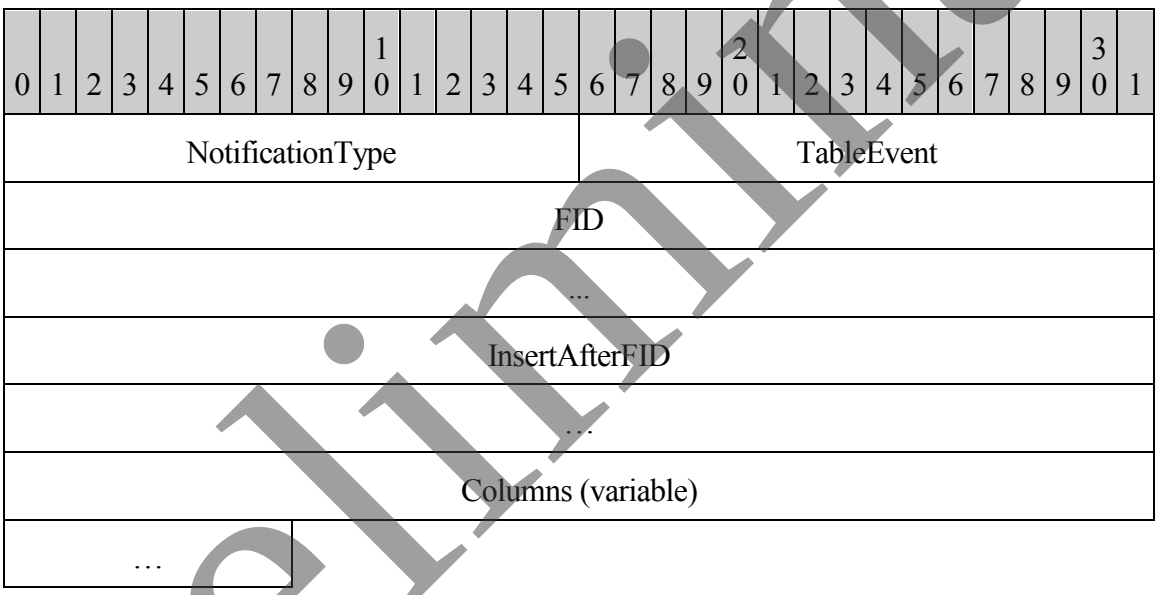
The combination of the FID**,** MID, and Instance makes a unique identifier for a row of a contents table. In order to update its view of a contents table when a new row notification arrives, the client SHOULD cache the rows it is currently displaying, and store the FID, MID, and Instance values with each row.

2.6.7.3 TABLE_ROW_MODIFIED Notifications

A client can use a TABLE_ROW_MODIFIED notification to directly update its row cache without pulling new data from the server.

There are two formats for TABLE_ROW_MODIFIED notifications: one for hierarchy tables and another for contents tables.

2.6.7.3.1 HierarchyRowModifiedNotification



NotificationType (2 bytes): Unsigned 16-bit integer. This value indicates the type of notification and MUST be set to 0x0100 by the server.

TableEvent (2 bytes): Unsigned 16-bit integer. This value indicates the type of table event and MUST be set to 0x0005 by the server.

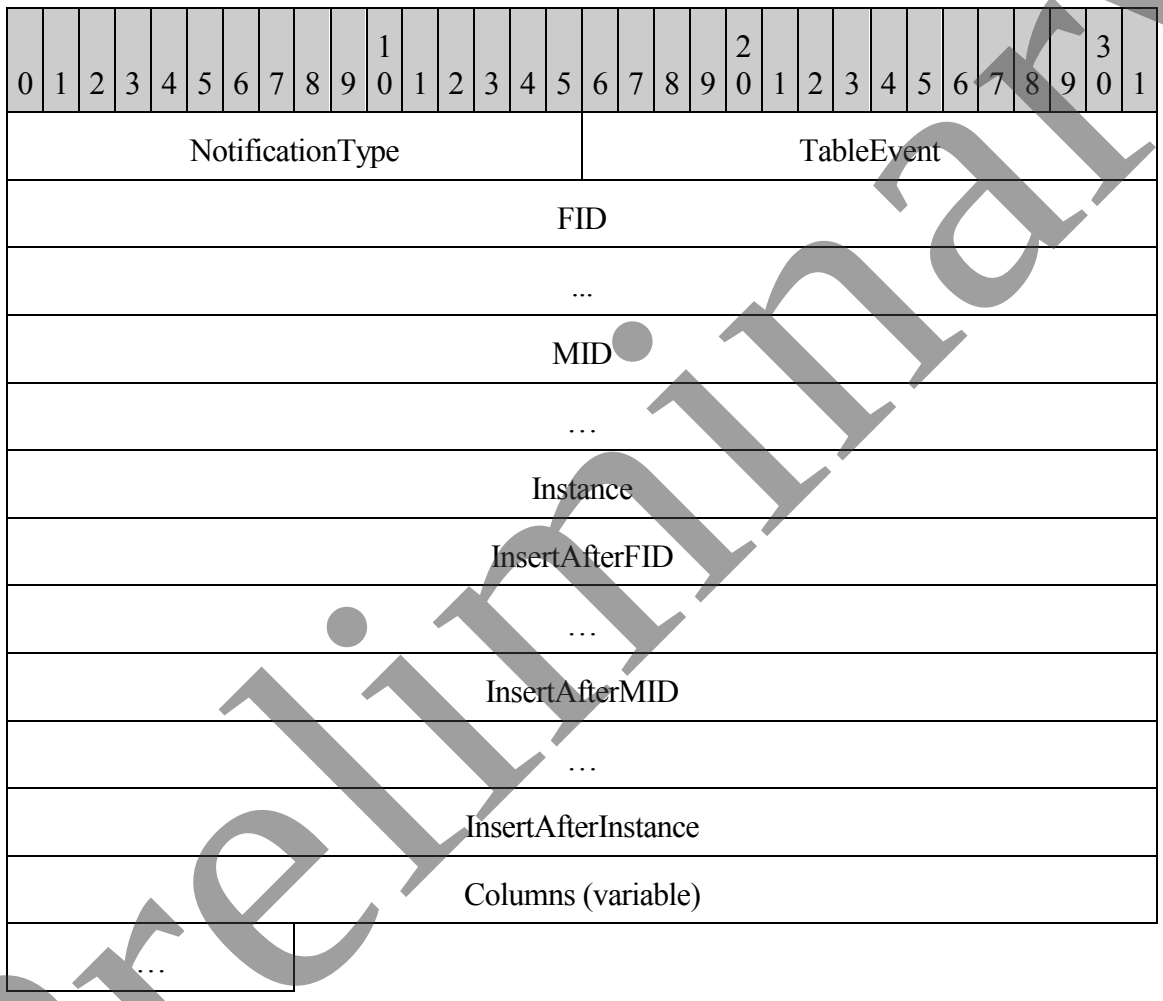
FID (8 bytes): FID structure. This value gives the FID of the row that was modified.

InsertAfterFID (8 bytes): FID structure. This value gives the FID of the folder row that the modified row SHOULD be moved after. If the modified folder row SHOULD be placed first, then this field MUST consist of 8 zero bytes.

Columns (variable): PropertyRow structure, as specified in section 2.10. This field gives the new values for the columns of the modified row.

The FID is the unique identifier for a row of a hierarchy table. In order to update its view of a hierarchy table when a new row notification arrives, a client SHOULD cache the rows it is currently displaying and store the FID with each row.

2.6.7.3.2 ContentsRowModifiedNotification



NotificationType (2 bytes): Unsigned 16-bit integer. This value indicates the type of notification and MUST be set to 0x8100 by the server.

TableEvent (2 bytes): Unsigned 16-bit integer. This value indicates the type of table event and MUST be set to 0x005 by the server.

FID (8 bytes): FID structure. This value gives the FID of the folder whose contents table has been modified.

MID (8 bytes): MID structure. When the table is categorized, this gives the category id of the modified row. In other cases, it gives the MID of the modified message object.

Instance (4 bytes): Unsigned 32-bit integer. When the table is categorized and one of the columns is multi-valued and specified as MultivalueInstance, this gives the instance number of the modified row. In other cases it will be 0.

InsertAfterFID (8 bytes): FID structure. This value gives the FID of the row that the modified row SHOULD be moved after. If the modified folder row SHOULD be placed first, then this field will consist of 8 zero bytes.

InsertAfterMID (8 bytes): MID structure. This value gives the MID (or category id) of the row that the modified row SHOULD be moved after. If the modified folder row SHOULD be placed first, then this field will consist of 8 zero bytes.

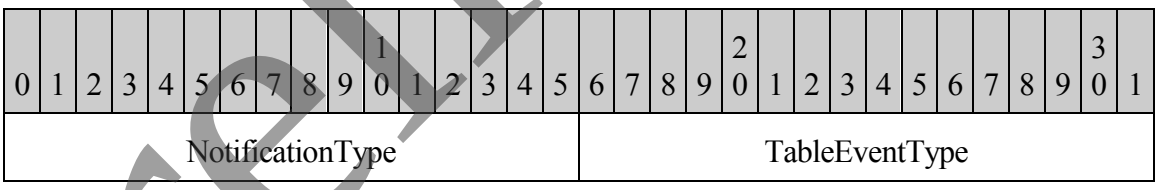
InsertAfterInstance (4 bytes): Unsigned 32-bit integer. This value gives the instance value (the value of PidTagInstanceKey) of the row after which the modified row SHOULD be inserted. If the modified folder row SHOULD be placed first, then this value will be zero.

Columns (variable): PropertyRow structure. This field gives the new values for the columns of the modified row.

The combination of the FID**,** MID, and Instance makes a unique identifier for a row of a contents table. In order to update its view of a contents table when a new row notification arrives, the client SHOULD cache the rows it is currently displaying, and store the FID, MID, and Instance values with each row.

2.6.7.4 TableChangedNotification

When a client receives a TABLE_CHANGED notification, it MUST discard its current rendering of the table and any cached data from the server, reread that data from the server, and recreate the rendering.

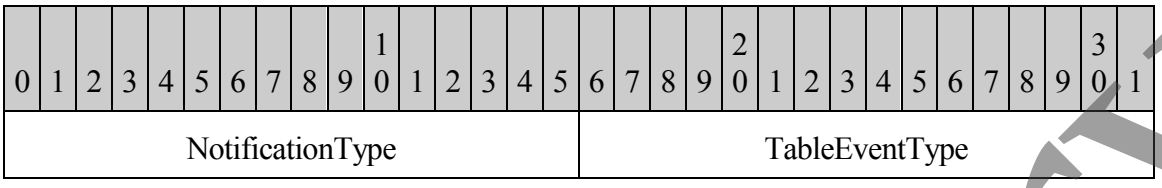


NotificationType (2 bytes): Unsigned 16-bit integer. This value indicates the type of notification and MUST be set by the server to either 0x0100 (for a hierarchy table) or 0x8100 (for a contents table).

TableEventType (2 bytes): An unsigned 16-bit integer. This value indicates the type of table event and MUST be set to 0x0001 by the server.

2.6.7.5 RestrictDoneNotification

The TABLE_RESTRICT_DONE notification tells a client that an asynchronous Restrict operation has completed on the server.

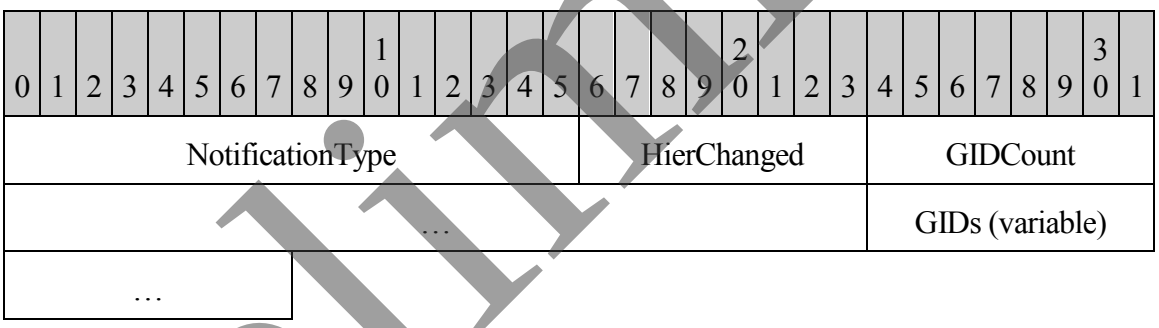


NotificationType (2 bytes): Unsigned 16-bit integer. This value indicates the type of notification and MUST be set to 0x0100 by the server.

TableEventType (2 bytes): Unsigned 16-bit integer. This value indicates the type of table event and MUST be set to 0x0007.

2.6.8 ICS Notification

An IcsNotification signals the client that one or more folders have changed on the server. The client SHOULD respond by resyncing the indicated folder or folders.



NotificationType (2 bytes): Unsigned 16-bit integer. This value indicates the type of notification and MUST be set to 0x0200 by the server.

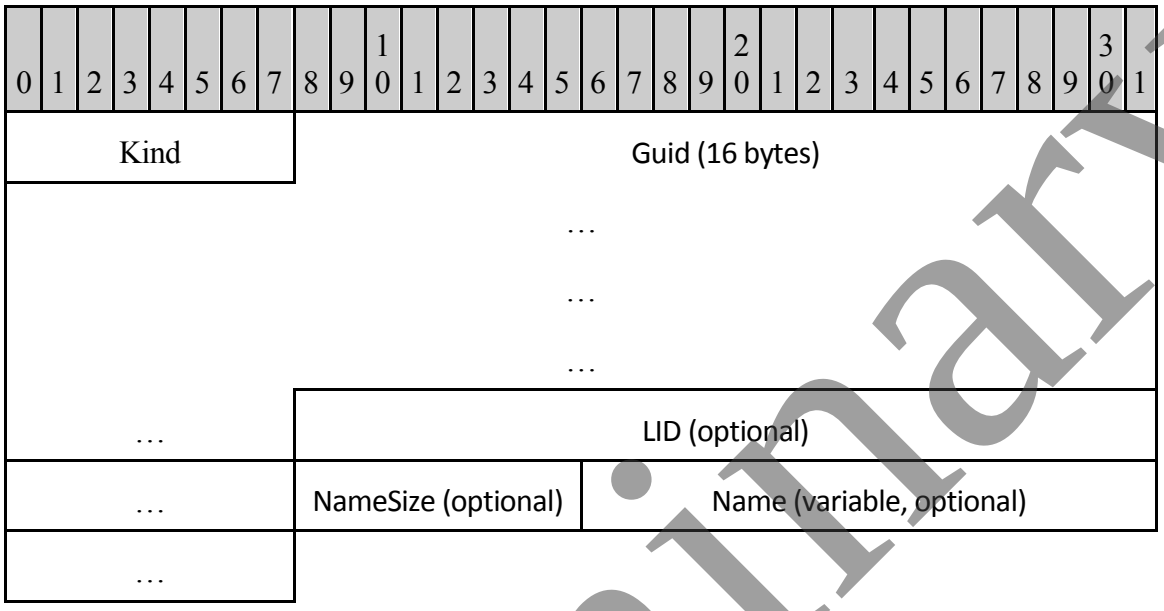
HierChanged (1 byte): Unsigned 8-bit integer. This value will be either TRUE (0x01) or FALSE (0x00) and indicates whether the folder hierarchy information has changed since the last ICS synchronization.

GIDCount (4 bytes): Unsigned 32-bit integer. This value indicates how many GID elements are present in **GIDs**.

GIDs (Variable): Array of GID structures. Each GID in this array specifies a folder whose contents have changed since the last ICS synchronization.

2.7 PropertyName

The PropertyName structure describes a named property. It is used in RopGetPropertyIdsFromNames and RopGetNamesFromPropertyIds requests.



Kind (1 byte): 0 if the property is identified by a LID, and 1 if the property is identified by a name.

Guid (16 bytes): The GUID that identifies the property set for the named property.
Note: Servers MUST NOT swap bytes for this GUID; it is treated as a FLATUID. Client code on big-endian systems MUST therefore place GUID fields in little-endian byte order in the request buffer.

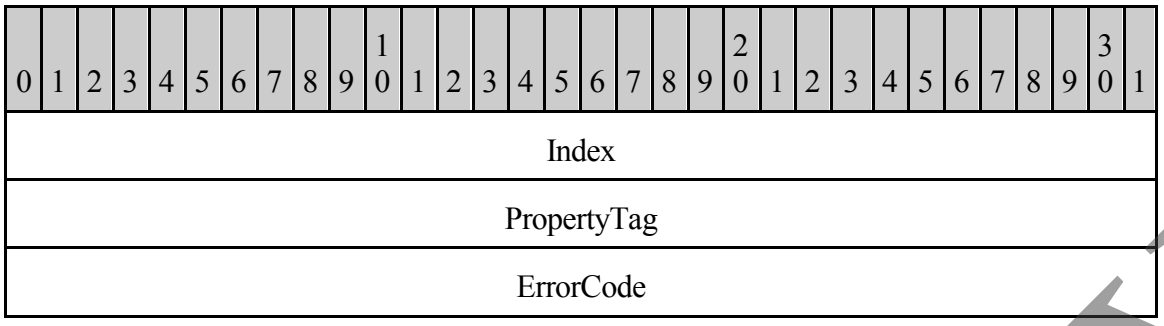
LID (4 bytes, optional): Present only if Kind=0. An unsigned 32-bit integer that identifies the named property within its property set.

NameSize (1 byte, optional): Present only if Kind=1. A single byte giving the number of bytes in the Name string that follows it.

Name (variable, optional): Present only if Kind=1. A Unicode (UTF-16) string, followed by two zero bytes as a null terminator, that identifies the property within its property set.

2.8 PropertyProblem

A PropertyProblem describes an error relating to an operation involving a property. It is often used in an array; see PropertyProblemArray.



Index (4 bytes): Unsigned 32-bit integer. This value specifies an index into an array of property tags.

PropertyTag (4 bytes): PropertyTag structure. This value specifies the property for which there was an error.

ErrorCode (4 bytes): Unsigned 32-bit integer. This value specifies the error that occurred when processing this property.

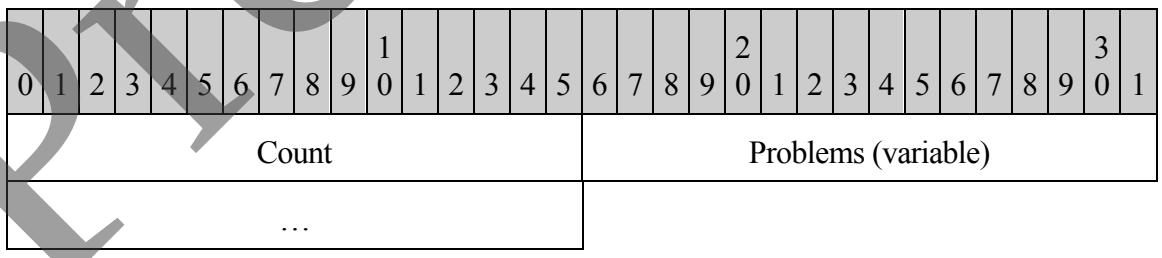
An array of PropertyProblem structures is returned from the following ROPs:

- RopDeleteProperties
- RopDeletePropertiesNoReplicate
- RopSetProperties
- RopSetPropertiesNoReplicate
- RopCopyProperties
- RopCopyTo

A PropertyProblem structure contains an error value that is a result of an operation attempting to modify or delete a property, as specified in Table 7. That property is identified by its PropertyTag, and also by its index in the property array passed to the request.

2.9 PropertyProblemArray

A PropertyProblemArray is a set of PropertyProblems that describe errors relating to an operation involving one or more properties.



Count (2 bytes): Unsigned 16-bit integer, specifying the number of PropertyProblems to follow.

Problems: 'Count' PropertyProblem structures, as specified in section **2.8.**.

2.10 PropertyRows

2.10.1 PropertyRow

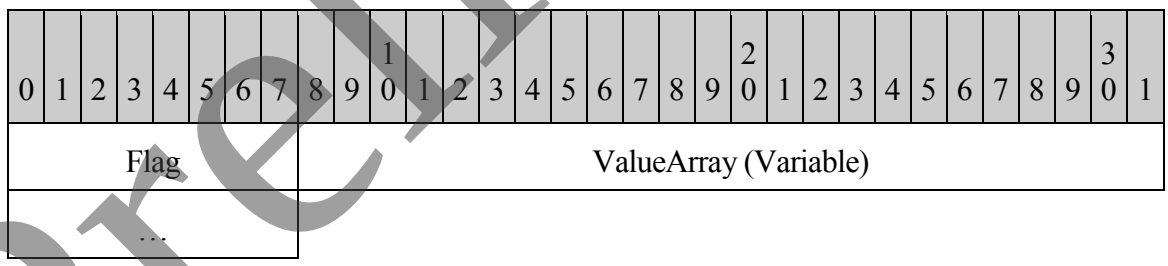
A PropertyRow structure is used to pass back a list of property values without including the property tag values that correspond to them. It is used to format property data returned to the client when the list of property tags is known in advance.

For instance, this data structure is used to format the response buffers of **RopGetPropertiesSpecific**, **RopFindRow**, and **RopGetReceiveFolderTable**. In addition, an array of PropertyRow structures makes up the key part of the PropertyRowSet structure returned in the response buffer for **RopQueryRows**. Finally, PropertyRow structures used in table notification structures to indicate the column values of a new added or modified row.

Since the property tags are not returned, clients SHOULD maintain state–perhaps from the request that receives a PropertyRow in its response, or perhaps from a previous request such as **ROPSetColumns** – in order to match property values from the row with their property tags.

There are two PropertyRow variants. A StandardPropertyRow contains no error values and no type data; it is simply a sequence of Property Values. A FlaggedPropertyRow MAY contain type data, if the request included PtypUnspecified for any property or column, and it MAY contain error values if a property value is missing or there was a problem retrieving the value. By examining the first byte of the property row, the client can identify the variant.

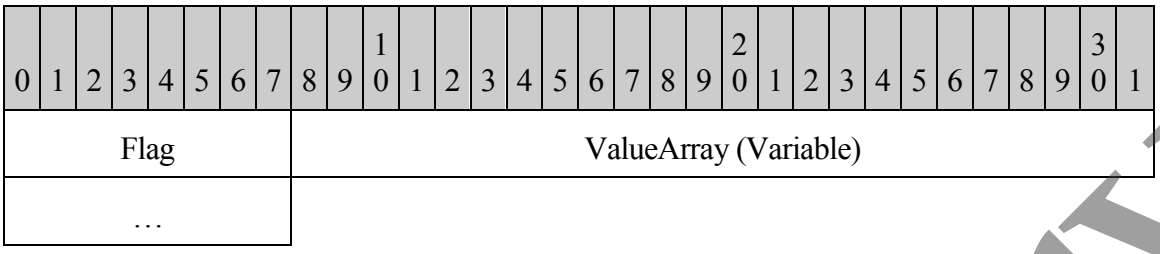
2.10.1.1 StandardPropertyRow



Flag (1 byte): Unsigned 8-bit integer. This value indicates whether all values are present and without error. This MUST be set to 0x00.

ValueArray (Variable): An array of variable-sized structures. At each position of the array, the structure will either be a PropertyValue structure (see section 2.13.2) if the type of the corresponding property tag was specified, or a TypedPropertyValue structure (see section 2.13.30) if the type of the corresponding property tag was PtypUnspecified.

2.10.1.2 FlaggedPropertyRow



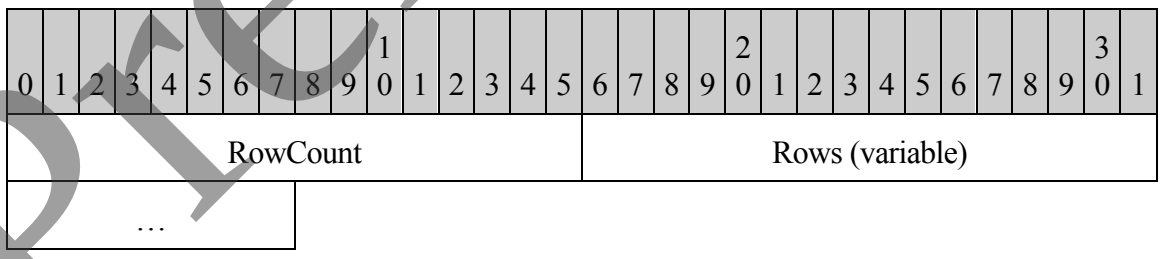
Flag (1 byte): Unsigned 8-bit integer. This value indicates whether all values are present and without error. This MUST be set to 0x01.

ValueArray (Variable): An array of variable-sized structures. At each position of the array, the structure will either be a FlaggedPropertyValue structure (see section 2.13.5) if the type of the corresponding property tag was previously specified, or a FlaggedPropertyValueWithTypeSpecified structure (see 0) if the type of the corresponding property tag was PtypUnspecified

2.10.2 PropertyRowSet

A PropertyRowSet is a counted series of PropertyRows. As for PropertyRow, the number of columns in each PropertyRow is not included in the PropertyRowSet. Clients SHOULD maintain state from a previous request, such as RopSetColumns, that includes the number of columns.

In table operations, such as in the response to a RopQueryRows request, servers SHOULD truncate long column values to a maximum of 255 bytes (for binary types) or 255 characters (for string types). Clients analyzing data returned from table operations SHOULD assume that if the length of such a value is exactly 255 bytes or characters, then the value of the same property obtained by opening the message and issuing a RopGetPropertiesSpecific request is likely to be larger.



RowCount (2 bytes): An unsigned 16-bit integer specifying the number of PropertyRows that follow.

Rows (variable size): A series of 'RowCount' PropertyRow structures.

2.10.3 RecipientRow

A RecipientRow structure represents a single recipient belonging to a message object. It is rather complex, but can be considered as a sequence of three different parts:

- A flags field indicating which of several standard properties are present
- Standard property values
- Arbitrary property values outside the standard set

This structure is used by several ROPs including:

- RopFlushRecipients
- RopReadRecipients
- RopOpenMessage
- RopOpenEmbeddedMessage
- RopRefreshCache

First, we specify the RecipientFlags field.

2.10.3.1 RecipientFlags

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	S	T	D	E	Type			O	Reserved							I	U	N													

R (1 bit): 1-bit flag (0x0080). If 1, a different transport is responsible for delivery to this recipient.

S (1 bit): 1-bit flag (0x0040). If 1, the Transmittable Display Name is the same as the Display Name.

T (1 bit): 1-bit flag (0x0020). If 1, the TransmittableDisplayName field is included.

D (1 bit): 1-bit flag (0x0010). If 1, the DisplayName field is included.

E (1 bit): 1-bit flag (0x0008). If 1, the EmailAddress field is included.

Type (3 bits): 3-bit enumeration (mask 0x0007). This enumeration specifies the type of address. The valid types are:

- NoType (0x0)
- X500DN (0x1)
- MsMail (0x2)
- SMTP (0x3)
- Fax (0x4)

- ProfessionalOfficeSystem (0x5)
- PersonalDistributionList1 (0x6)
- PersonalDistributionList2 (0x7)

O (1 bit): 1-bit flag (0x8000). If 1,this recipient has a non-standard address type and the AddressType field is included.

Reserved (4 bits): (mask 0x0078) The server MUST set this to 0.

I (1 bit): 1-bit flag (0x0400). If 1, the SimpleDisplayName is included.

U (1 bit): 1-bit flag (0x0200). If 1, the associated string properties are in Unicode with a 2-byte null terminator; if 0, string properties are MBCS with a single null terminator, in the codepage sent to the server in **EcDoConnect** (as specified in [MS-OXCRPC]).

N (1 bit): 1-bit flag (0x0100). This flag specifies that the recipient does not support receiving rich text messages.

2.10.3.2 RecipientRow

<table>
<tr><th>0</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>10</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>20</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>30</th><th>1</th></tr>
<tr><td colspan="16">RecipientFlags</td><td colspan="8">AddressPrefixUsed (optional)</td><td colspan="8">DisplayType (optional)</td></tr>
<tr><td colspan="32">X500DN (variable, optional)</td></tr>
<tr><td colspan="16">EntryIdSize (optional)</td><td colspan="16">EntryId (variable, optional)</td></tr>
<tr><td colspan="8">…</td><td colspan="16">SearchKeySize (optional)</td><td colspan="8">SearchKey (variable, optional)</td></tr>
<tr><td colspan="8">…</td><td colspan="24">AddressType (variable, optional)</td></tr>
<tr><td colspan="8">…</td><td colspan="24">EmailAddress (variable, optional)</td></tr>
<tr><td colspan="8">…</td><td colspan="24">DisplayName (variable, optional)</td></tr>
<tr><td colspan="8">…</td><td colspan="24">SimpleDisplayName (variable, optional)</td></tr>
<tr><td colspan="8">…</td><td colspan="24">TransmittableDisplayName (variable, optional)</td></tr>
<tr><td colspan="8">…</td><td colspan="16">RecipientColumnCount</td><td colspan="8">RecipientProperties (variable)</td></tr>
<tr><td colspan="8">…</td></tr>
</table>

RecipientFlags (2 bytes): RecipientFlags structure. The format of this structure is defined in section 2.10.3.1. This value specifies the type of recipient and which standard properties are included.

AddressPrefixUsed (1 byte, optional): Unsigned 8-bit integer. This field MUST be present when the Type field of the RecipientFlags field is set to X500DN (0x1) and MUST NOT be present otherwise. This value specifies the amount of the Address Prefix is used for this X500 DN.

DisplayType (1 byte, optional): 8-bit enumeration. This field MUST be present when the Type field of the RecipientFlags field is set to X500DN (0x1) and MUST NOT be present otherwise. This value specifies the display type of this address.

X500DN (variable, optional): Null-terminated ASCII string. This field MUST be present when the Type field of the RecipientFlags field is set to X500DN (0x1) and MUST NOT be present otherwise. This value specifies the X500 DN of this recipient.

EntryIdSize (2 bytes, optional): Unsigned 16-bit integer. This field MUST be present when the Type field of the RecipientFlags field is set to PersonalDistributionList1 (0x6) or PersonalDistributionList2 (0x7). This field MUST NOT be present otherwise. This value specifies the size of the EntryId field.

EntryId (variable, optional): Array of bytes. This field MUST be present when the Type field of the RecipientFlags field is set to PersonalDistributionList1 (0x6) or PersonalDistributionList2 (0x7). This field MUST NOT be present otherwise. The number of bytes in this field MUST be the same as specified in the EntryIdSize field. This array specifies the Address Book EntryId of the Distribution List.

SearchKeySize (2 bytes, optional): Unsigned 16-bit integer. This field MUST be present when the Type field of the RecipientFlags field is set to PersonalDistributionList1 (0x6) or PersonalDistributionList2 (0x7). This field MUST NOT be present otherwise. This value specifies the size of the SearchKey field.

SearchKey (variable, optional): Array of bytes. This field MUST be present when the Type field of the RecipientFlags field is set to PersonalDistributionList1 (0x6) or PersonalDistributionList2 (0x7). This field MUST NOT be present otherwise. The number of bytes in this field MUST be the same as specified in the SearchKeySize field. This array specifies the Search Key of the Distribution List.

AddressType (variable, optional): Null-terminated ASCII string. This field MUST be present when the Type field of the RecipientFlags field is set to NoType (0x0) and the O flag of the RecipientsFlags field is set. This field MUST NOT be present otherwise. This string specifies the address type of the recipient.

EmailAddress (variable, optional): Null-terminated string. This field MUST be present when the E flag of the RecipientsFlags field is set and MUST NOT be present otherwise. This field MUST be specified in Unicode characters if the U flag of the RecipientsFlags field is set and 8-bit character set otherwise. This string specifies the Email Address of the recipient.

DisplayName (variable, optional): Null-terminated string. This field MUST be present when the D flag of the RecipientsFlags field is set and MUST NOT be present otherwise. This field MUST be specified in Unicode characters if the U flag of the RecipientsFlags field is set and 8-bit character set otherwise. This string specifies the Email Address of the recipient.

SimpleDisplayName (variable, optional): Null-terminated string. This field MUST be present when the I flag of the RecipientsFlags field is set and MUST NOT be present otherwise. This field MUST be specified in Unicode characters if the U flag of the RecipientsFlags field is set and 8-bit character set otherwise. This string specifies the Email Address of the recipient.

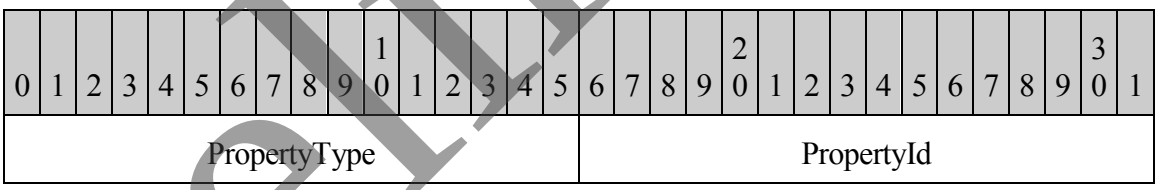
TransmittableDisplayName (variable, optional): Null-terminated string. This field MUST be present when the T flag of the RecipientsFlags field is set and MUST NOT be present otherwise. This field MUST be specified in Unicode characters if the U flag of the RecipientsFlags field is set and 8-bit character set otherwise. This string specifies the Email Address of the recipient.

RecipientColumnCount (2 bytes): Unsigned 16-bit integer. This value specifies the number of columns from the RecipientColumns field that are included in RecipientProperties**.**

RecipientProperties (variable): PropertyRow structures. The format of the PropertyRow structure is defined in section 2.10 and the columns used for this row are those specified in RecipientProperties.

2.11 PropertyTag, PropertyId

A property tag both identifies a property and gives the data type of its value. Please refer to [MS-OXPROPS] and other sections of this document for further information about PropertyIds and PropertyValues.

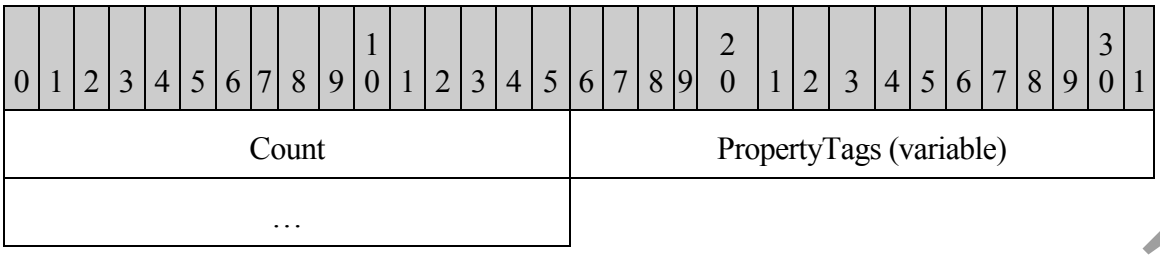


PropertyType (2 bytes): A 16-bit unsigned integer that identifies the data type of the property value, as specified by the table in section 2.13.3.

PropertyId (2 bytes): A 16-bit unsigned integer that identifies the property.

2.12 PropertyTagArray

A PropertyTagArray is simply a counted set of property tags, laid out as follows.



Count (2 bytes): Unsigned 16-bit integer, specifying the number of property tags to follow.

PropertyTags: 'Count' unsigned 32-bit integers representing property tags.

2.13 Property Values

There are a variety of structures used for conveying the value of a property to and from the server. Some variants contain only the value, because the usage context dictates the type. Other variants include the type, or the full property tag. Still others include an indication of whether an error occurred.

2.13.1 Property Value Types

For all variants, the structure of a property value is the same and is specified by the property value type, whether or not the property value type is actually encoded in the buffer. The following table lists both the property value type identifiers and the format of the property values themselves.

There is one variation in the width of count fields. In the context of ROP buffers, such as RopGetPropertiesSpecific, byte counts for PtypBinary property values and value counts for all PtypMultiple property values are 16 bytes wide. But in the context of Extended Rules, as specified in [MS-OXRULE], byte counts and property value counts are 32 bits wide (for example, COUNT in the table below represents a 32-bit integer). Such count fields have a width designation of COUNT, rather than an explicit 1-byte width, throughout section 2.13.

In the context of a table operation, properties are referred to as columns. The format of property identifiers, types, and values in table operations such as RopQueryRows is the same as in property operations such as RopGetPropertiesSpecific.

Property value types are presented in the following table:

Table 10 Property Value Types

Property type name	Property type value	Property type specification	Alternate names
PtypInteger16	0x0002, %x02.00	2 bytes, a 16-bit integer [MS-DTYP]: INT16	PT_SHORT, PT_I2

PtypInteger32	0x0003, %x03.00	4 bytes, a 32-bit integer [MS-DTYP]: INT32	PT_LONG, PT_I4
PtypFloating32	0x0004, %x04.00	4 bytes, a 32-bit floating point number [MS-DTYP]: FLOAT	PT_FLOAT, PT_R4
PtypFloating64	0x0005, %x05.00	8 bytes, a 64-bit floating point number [MS-DTYP]: DOUBLE	PT_DOUBLE, PT_R8
PtypCurrency	0x0006, %x06.00	8 bytes, a 64-bit signed, scaled integer representation of a decimal currency value, with 4 places to the right of the decimal point [MS-DTYP]: LONGLONG [MS-OAUT]: CURRENCY	PT_CURRENCY
PtypFloatingTime	0x0007, %x07.00	8 bytes, a 64-bit floating point number in which the whole number part represents the number of days since December 30, 1899, and the fractional part represents the fraction of a day since midnight [MS-DTYP]: DOUBLE [MS-OAUT]: DATE	PT_APPTIME
PtypErrorCode	0x000A, %x0A.00	4 bytes, a 32-bit integer encoding error information as specified in 2.4.1	PT_ERROR
PtypBoolean	0x000B, %x0B.00	1 byte, restricted to 1 or 0 [MS-DTYP]: BOOLEAN	PT_BOOLEAN
PtypInteger64	0x0014, %x14.00	8 bytes, a 64-bit integer [MS-DTYP]: LONGLONG	PT_LONGLONG, PT_I8
PtypString	0x001F, %x1F.00	Variable size, a string of Unicode characters in UTF-16LE encoding with terminating null character (2 bytes of zero)	PT_UNICODE

PtypString8	0x001E, %z1E.00	Variable size, a string of multi-byte characters in externally specified encoding with terminating null character (single 0 byte)	PT_STRING8
PtypTime	0x0040, %x40.00	8 bytes, a 64-bit integer representing the number of 100-nanosecond intervals since January 1, 1601 [MS-DTYP]: FILETIME	PT_SYSTIME
PtypGuid	0x0048, %x48.00	16 bytes, a GUID with Data1, Data2, and Data3 fields in little-endian format [MS-DTYP]: GUID	PT_CLSID
PtypServerId	0x00FB, %xFB.00	Variable size, a 16-bit count followed a structure specified in section 2.13.1.3.	PT_SVREID
PtypRestriction	0x00FD, %xFD.00	Variable size, a byte array representing one or more Restriction structures as specified in section 2.14	PT_SRESTRICT
PtypRuleAction	0x00FE, %xFE.00	Variable size, a 16-bit count of actions (not bytes) followed by that many Rule Action structures, as specified in [MS-OXORULE]	PT_ACTIONS
PtypBinary	0x0102, %x02.01	Variable size, a COUNT followed by that many bytes	PT_BINARY
PtypMultipleInteger16	0x1002, %x02.10	Variable size, a COUNT followed by that many PtypInteger16 values	PT_MV_SHORT, PT_MV_I2
PtypMultipleInteger32	0x1003, %x03.10	Variable size, a COUNT followed by that many PtypInteger32 values	PT_MV_LONG, PT_MV_I4

PtypMultipleFloating32	0x1004, %x04.10	Variable size, a COUNT followed by that many PtypFloating32 values	PT_MV_FLOAT, PT_MV_R4
PtypMultipleFloating64	0x1005, %x05.10	Variable size, a COUNT followed by that many PtypFloating64 values	PT_MV_DOUBLE, PT_MV_R8
PtypMultipleCurrency	0x1006, %x06.10	Variable size, a COUNT followed by that many PtypCurrency values	PT_MV_CURRENCY
PtypMultipleFloatingTime	0x1007, %x07.10	Variable size, a COUNT followed by that many PtypFloatingtime values	PT_MV_APPTIME
PtypMultipleInteger64	0x1014, %x14.10	Variable size, a COUNT followed by that many PtypInteger64 values	PT_MV_I8, PT_MV_LONGLONG
PtypMultipleString	0x101F, %x1F.10	Variable size, a COUNT followed by that PtypString values	PT_MV_UNICODE
PtypMultipleString8	0x101E, %x1E.10	Variable size, a COUNT followed by that many PtypString8 values	PT_MV_STRING8
PtypMultipleTime	0x1040, %x40.10	Variable size, a COUNT followed by that many PtypTime values	PT_MV_SYSTIME
PtypMultipleGuid	0x1048, %x48.10	Variable size, a COUNT followed by that many PtypGuid values	PT_MV_CLSID
PtypMultipleBinary	0x1102, %x02.11	Variable size, a COUNT followed by that many PtypBinary values	PT_MV_BINARY

PtypUnspecified	0x0000, %x00.00	Any: this property type value matches any type; a server MUST return the actual type in its response. Servers MUST NOT return this type in response to a client request other than NspiGetIDsFromNames or RopGetPropertyIdsFromNames.	PT_UNSPECIFIED
PtypNull	0x0001, %x01.00	None: This property is a placeholder	PT_NULL
PtypObject or PtypEmbeddedTable	0x000D, %x0d.00	The property value is a COM object, as specified in section 2.13.1.4	PT_OBJECT

2.13.1.1 String Property Values

Clients SHOULD work with PtypString and PtypMultipleString properties in Unicode format. When working with strings in Unicode format, string data MUST be encoded as UTF-16LE, and property data types MUST be specified as 0x001F (PtypString) and 0x101F (PtypMultipleString).

Clients MAY, instead, work with PtypString8 and PtypMultipleString8 properties in a specific 8-bit or multibyte codepage. In this case, property data types MUST be formatted as 0x001E (PtypString8) and 0x101E (PtypMultipleString8).

In requests sent to a store server, the codepage of PtypString8 strings MUST match the codepage sent to the server in **EcDoConnect** or similar RPC, as specified in [MS-OXCRPC]. Address book server rules for working with PtypString8 properties are somewhat more involved, and are specified in [MS-NSPI].

2.13.1.2 Multi-Valued Property Value Instances

When working with multi-valued columns in the context of table operations, clients MAY set the 0x2000 (MultivalueInstance, %x00.20) flag bit in the column's PropertyType field to indicate that the multi-valued column is to be treated as individual values. The MultivalueInstance flag MUST NOT be set for any column that does not also set the 0x1000 (Multivalue) bit in its property type. All PtypMultiple types in Table 10 set the 0x1000 bit.

The MultivalueInstance flag causes table operations to treat multi-valued columns as if they were multiple instances of a single-valued column. Please refer to [MS-OXCTABL] for table ROP specifications.

2.13.1.3 The PtypServerId Type

This property type encapsulates a message database EntryID. A ServerId identifies either a folder object or a message object.

Ours (1 byte): 0x01 indicates the remaining bytes conform to this structure; 0x00 indicates this is a client-defined value, and has whatever size and structure the client has defined.

FolderId (8 bytes): A FID, as specified in section 2.2.2.1, identifying a folder.

MessageId (8 bytes): A MID, as specified in section 2.2.2.2, identifying a message in the folder identified by FolderId. If the object is a folder, this field MUST be all zeros.

Instance (4 bytes): A 32-bit unsigned instance number within an array of ServerIds to compare against. This field is used only for searches against multi-value properties and MUST be zero in any other context.

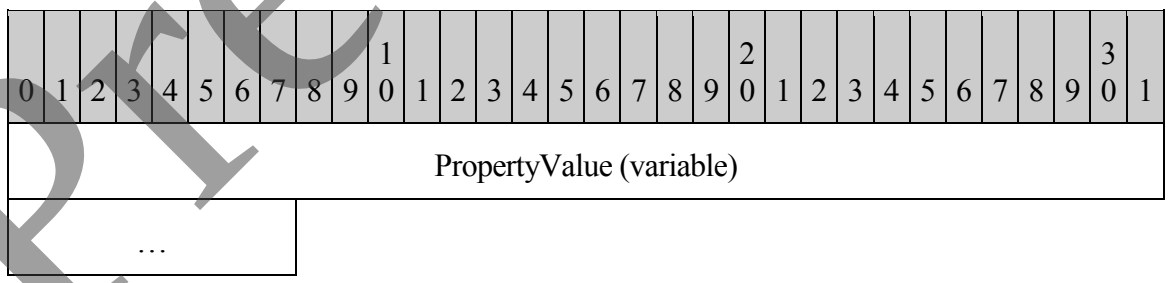
2.13.1.4 PtypObject and PtypEmbeddedTable

Store and address book servers treat this property type somewhat differently, but in both cases a property of this type represents a complex structure. Access to properties of this type requires the server to construct an object, and the client to issue requests similar to those used for top-level objects.

- Store servers do not allow access to properties of type PtypObject through RopGetPropertiesSpecific or RopGetPropertiesAll. Instead, properties of this type MUST be accessed with RopOpenStream or RopOpenEmbeddedMessage requests, as specified in [MS-OXCROPS].
- Address book servers use PtypEmbeddedTable to designate properties whose value is a table, for example, the members of a distribution list. The necessary methods are specified in [MS-NSPI].

2.13.2 PropertyValue

The PropertyValue structure simply specifies the value of the property. It contains no information about the property type or id.

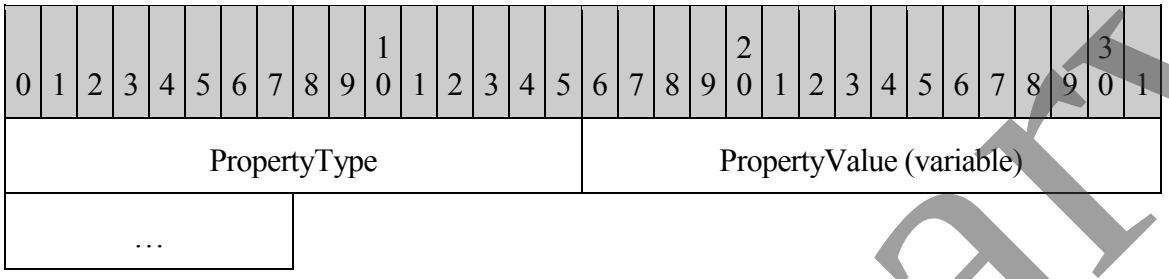


PropertyValue (variable size): The size varies depending on the property type which can be understood from the usage context. All numeric values are in little-endian format. For multi-

valued types, the first element in the ROP buffer is a 16-bit integer specifying the number of entries.

2.13.3 TypedPropertyValue

The TypedPropertyValue structure includes the property type with the value of the property.



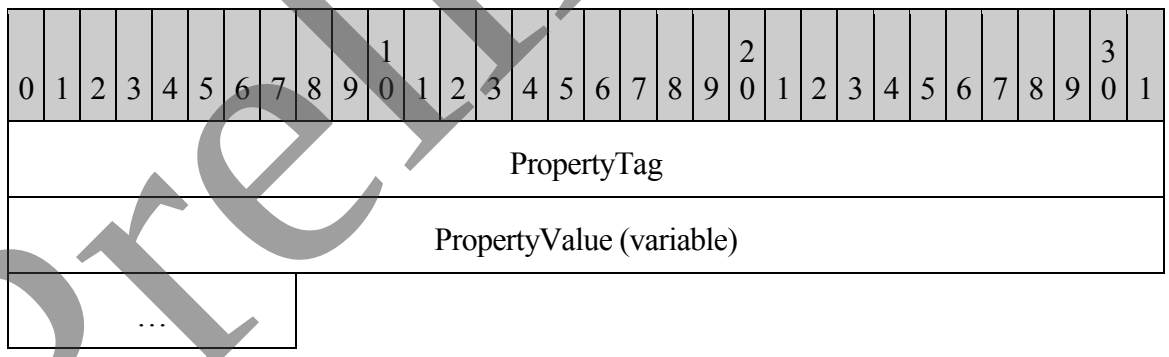
PropertyType (2 bytes): A 16-bit unsigned integer that specifies the data type of the property value, according to the table in section 2.13.1.

PropertyValue (variable size): A PropertyValue structure as specified in section 2.13.2. The value MUST be compatible with the value of the PropertyType field.

2.13.4 TaggedPropertyValue

As a rule, property tags are not specified explicitly in ROP buffers. To save space, property tags are specified implicitly by a previous operation and only the property values are put in the buffer. But under some circumstances a TaggedPropertyValue is used to explicitly include the property type and ID in the buffer

.

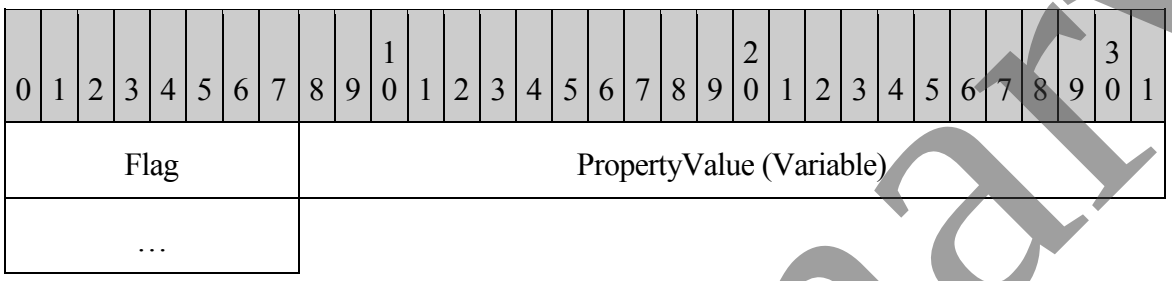


PropertyTag (32 bits): A PropertyTag structure giving the PropertyId and PropertyType for the property.

PropertyValue (variable size): A PropertyValue structure specifying the value of the property. Its syntax is specified by the PropertyType field of the tag, and its semantics by the PropertyId field of the tag.

2.13.5 FlaggedPropertyValue

This variant includes a flag to indicate whether the value was successfully retrieved or not. Error conditions include a missing property or a failure at the server.



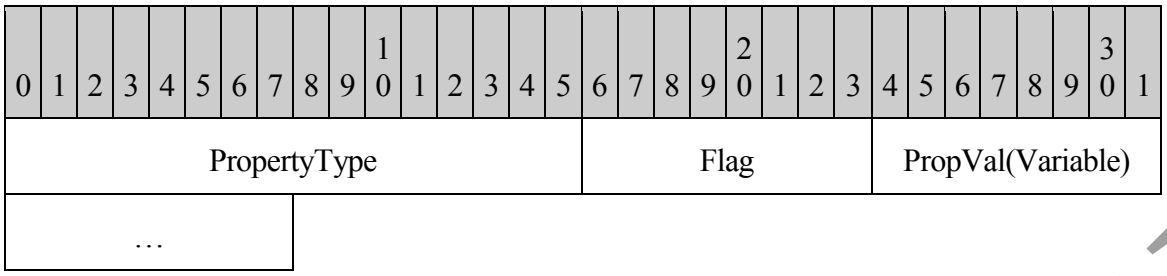
Flag (1 byte): An 8-bit unsigned integer. This flag MUST be set one of three possible values: 0x0, 0x1, or 0xA, which determines what is conveyed in the **PropertyValue** field. The following table summarizes the meanings of these three values:

Flag value	What it implies about the **PropertyValue** field
0x0	The **PropertyValue** field will be PropertyValue structure containing a value compatible with the property type implied the context.
0x1	The **PropertyValue** field is not present.
0xA	The **PropertyValue** field will be a PropertyValue structure containing an unsigned 32-bit integer. This value SHOULD be interpreted as a property error code (see section 2.4.2) indicating why the property value is not present.

PropertyValue (variable size): A PropertyValue structure (see section 2.13.2) unless the Flag field is 0x1.

2.13.6 FlaggedPropertyValueWithType

This variant includes both the property type and a flag giving more information about the property value.



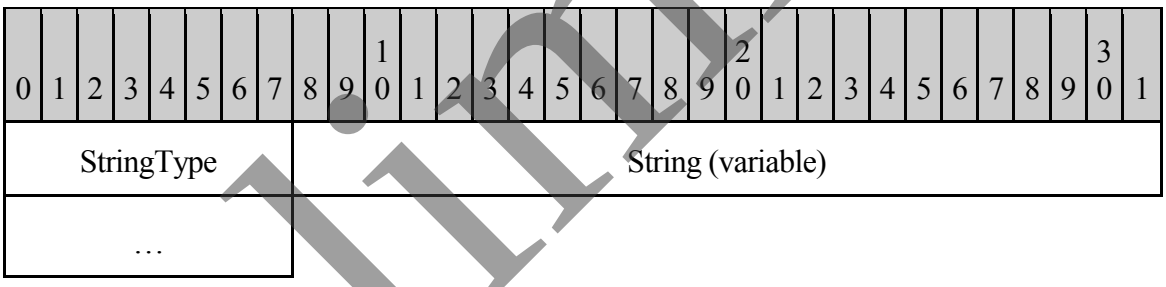
PropertyType (2 bytes): A 16-bit unsigned integer that specifies the data type of the property value, according to the table immediately below.

Flag (1 byte): An 8-bit unsigned integer. This flag MUST be set one of three possible values: 0x0, 0x1, or 0xA, which determines what is conveyed in the PropertyValue field. Refer to the table in section 2.13.5 for the interpretation of this flag.

PropertyValue (variable size): A PropertyValue structure (see section 2.13.2). The value MUST be compatible with the value of the PropertyType field.

2.13.7 TypedString

A TypedString is used in certain ROPs in order to compact the string representation on the wire as much as possible.



StringType (1 byte): 8-bit enumeration. The value MUST be one of the following:

- 0x00: There is no string present.
- 0x01: The string is empty.
- 0x02: Null-terminated 8-bit character string. The null terminator is one zero byte.
- 0x03: Null-terminated Reduced Unicode character string. The null terminator is one zero byte.
- 0x04: Null-terminated Unicode character string. The null terminator is 2 zero bytes.

String (variable, optional): If the StringType field is set to 0x02, 0x03, or 0x04, then this field MUST be present and in the format specified by the Type field. Otherwise, this field MUST NOT be present.

To produce a Reduced Unicode string from an original Unicode string, the server MUST first scan the original Unicode string and determine that every character has a value less than 0x100; in other words, that the high-order byte of every character, including the null terminator, is zero. It MUST then produce a Reduced Unicode string that is exactly half the size of the original Unicode string by omitting all the high-order zero bytes, including that of the null terminator.

To reproduce the original Unicode string from a Reduced Unicode string, the server MUST insert a zero byte after each byte of the Reduced Unicode string, doubling its size.

2.14 Restrictions

Restrictions describe a filter for limiting the view of a table to particular set of rows. This filter represents a Boolean expression that is evaluated against each item of the table. The item will be included as a row of the restricted table if and only if the value of the Boolean expression evaluates to TRUE.

Restrictions are sent to the server with the RopFindRow, RopRestrict, RopSetSearchCriteria, and RopSynchronizationConfigure requests, and are returned from the RopGetSetSearchCriteria request.

There are 12 different restriction packet formats: Six of them (AndRestriction, OrRestriction, NotRestriction, SubRestriction, CommentRestriction, and CountRestriction) are used to construct more complicated restrictions from one or more simpler ones. The other six types (ContentRestriction, PropertyRestriction, ComparePropertiesRestriction, BitMaskRestriction, SizeRestriction, and ExistRestriction) specify specific tests based on the properties of an item.

While the packet formats differ, the first 8 bits always stores RestrictType, an unsigned byte value specifying the type of restriction. The possible values for RestrictType are are presented in the following table:

Table 11 Restrict Type Values

RestrictType	Hexadecimal value (alternate name)	Description
AndRestriction	0x00 (RES_AND)	Logical **AND** operation applied to a list of subrestrictions.
OrRestriction	0x01 (RES_OR)	Logical **OR** operation applied to a list of subrestrictions.

RestrictType	Hexadecimal value (alternate name)	Description
NotRestriction	0x02 (RES_NOT)	Logical **NOT** applied to a subrestriction
ContentRestriction	0x03 (RES_CONTENT)	Search a property value for specific content.
PropertyRestriction	0x04 (RES_PROPERTY)	Compare a property value to a particular value.
ComparePropertiesRestriction	0x05 RES_COMPAREPROPS	Compare the values of two properties.
BitMaskRestriction	0x06 (RES_BITMASK)	Perform bitwise **AND** of a property value with a mask and compare to zero
SizeRestriction	0x07 (RES_SIZE)	Compare the size of a property value to a particular figure.
ExistRestriction	0x08 (RES_EXIST)	Test whether a property has a value.
SubObjectRestriction	0x09 (RES_SUBRESTRICTION)	Test whether any row of a message's attachment or recipient table satisfies a subrestriction.
CommentRestriction	0x0A (RES_COMMENT)	Associates a comment with a subrestriction.
CountRestriction	0x0B (RES_COUNT)	Limits the number of matches returned from a subrestriction.

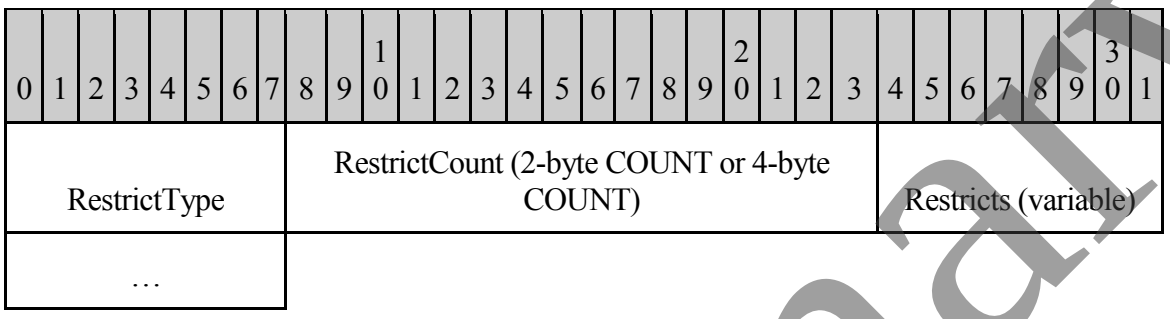
The subsections which follow describe each packet format.

There is one variation in the way Restriction structures are serialized. In the context of ROP buffers, such as RopRestrict or RopSetSearchCriteria, all count fields (such as the number of subrestrictions of an AndRestriction) are 16 bits wide. But in the context of Extended Rules,

as specified in [MS-OXRULE], these counts are 32 bits wide. Such fields are identified as COUNT fields throughout section 2.14.

2.14.1 AndRestriction

The **AndRestriction** structure describes an AND restriction, which is used to join a group of restrictions using a logical AND operation.



RestrictType (1 byte): Unsigned 8-bit integer. This value indicates the type of restriction and MUST be set to 0x00.

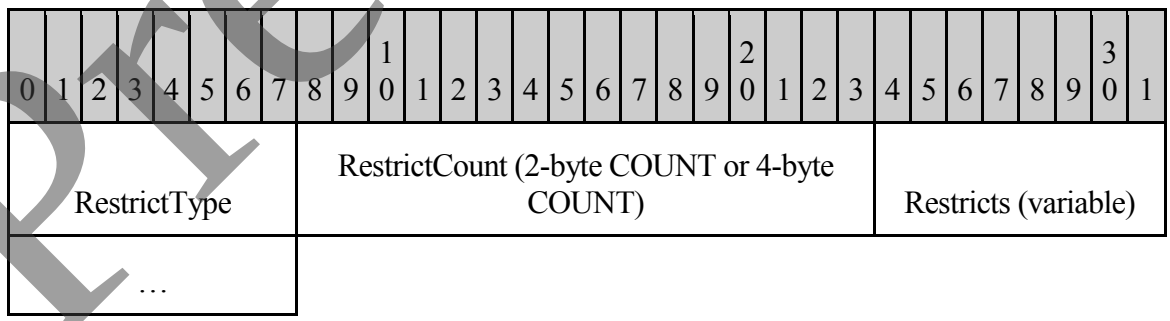
RestrictCount (COUNT): This value specifies how many Restriction structures are present in **Restricts**.

Restricts (variable): Array of Restriction structures. This field MUST contain **RestrictCount** structures.

The result of an AndRestriction is TRUE if all of its child restrictions evaluate to TRUE, and FALSE if any child restriction evaluates to FALSE.

2.14.2 OrRestriction

The OrRestriction structure describes an OR restriction, which is used to join a group of restrictions using a logical OR operation.



RestrictType (1 byte): Unsigned 8-bit integer. This value indicates the type of restriction and MUST be set to 0x01.

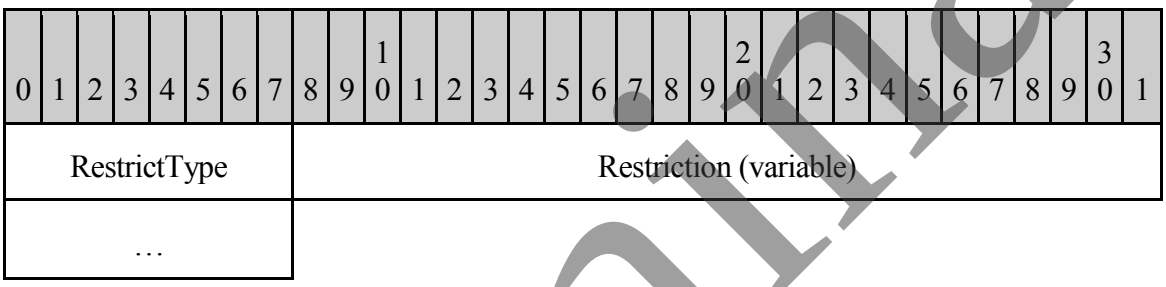
RestrictCount (COUNT): This value specifies how many Restriction structures are present in **Restricts**.

Restricts (variable): Array of Restriction structures. This field MUST contain **RestrictCount** structures.

The result of an **OrRestriction** is TRUE if at least one of its child restrictions evaluates to TRUE, and FALSE if all child restrictions evaluate to FALSE.

2.14.3 NotRestriction

The NotRestriction structure describes a NOT restriction, which is used to apply a logical NOT operation to a single restriction.



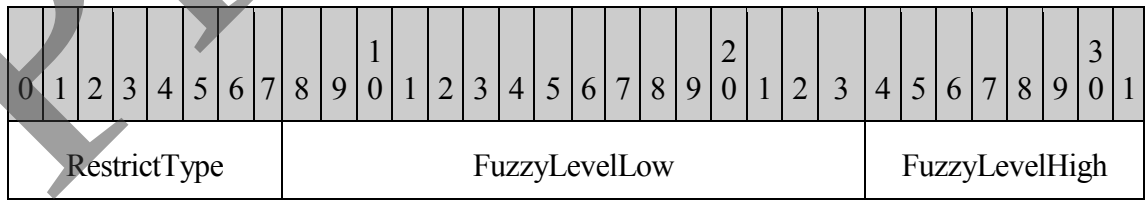
RestrictType (1 byte): Unsigned 8-bit integer. This value indicates the type of restriction and MUST be set to 0x02.

Restriction (variable): A restriction structure. This value specifies the restriction the logical NOT SHOULD be applied to.

The result of a NotRestriction is TRUE if the child restriction evaluates to FALSE, and FALSE if the child restriction evaluates to TRUE.

2.14.4 ContentRestriction

The ContentRestriction structure describes a content restriction, which is used to limit a table view to only those rows that include a column with contents matching a search string.



…	PropertyTag
…	TaggedValue (variable)
…	

RestrictType (1 byte): Unsigned 8-bit integer. This value indicates the type of restriction and MUST be set to 0x03.

FuzzyLevelLow (2 bytes): Unsigned 16-bit integer. This field specifies the level of precision that the server SHOULD enforce when checking for a match against a ContentRestriction. The lower 16 bits of the ulFuzzyLevel member apply to both binary and string properties and MUST be set to one of the following values:

FuzzyLevelLow value	Description
0x0000 FL_FULLSTRING	The value stored in TaggedValue and the value of the column PropertyTag MUST match in their entirety.
0x0001 FL_SUBSTRING	The value stored in TaggedValue MUST match some portion of the value of the column PropertyTag.
0x0002 FL_PREFIX	The value stored in TaggedValue MUST match a starting portion of the value of the column PropertyTag.

FuzzyLevelHigh (2 bytes): This field applies only to string valued properties and can be set to the following bit values in any combination:

FuzzyLevelHigh values	Description
0x0001 FL_IGNORECASE	The comparison SHOULD be made without considering case.
0x0002 FL_IGNORENONSPACE	The comparison SHOULD ignore Unicode-defined nonspacing characters such as diacritical marks.

0x0004 FL_LOOSE	The comparison SHOULD result in a match whenever possible, ignoring case and nonspacing characters.

PropertyTag (4 bytes): Unsigned 32 bit integer. This value indicates the property tag of the column that whose value MUST be matched against the value specified by the TaggedValue field. The type of this property MUST NOT be multi-valued.

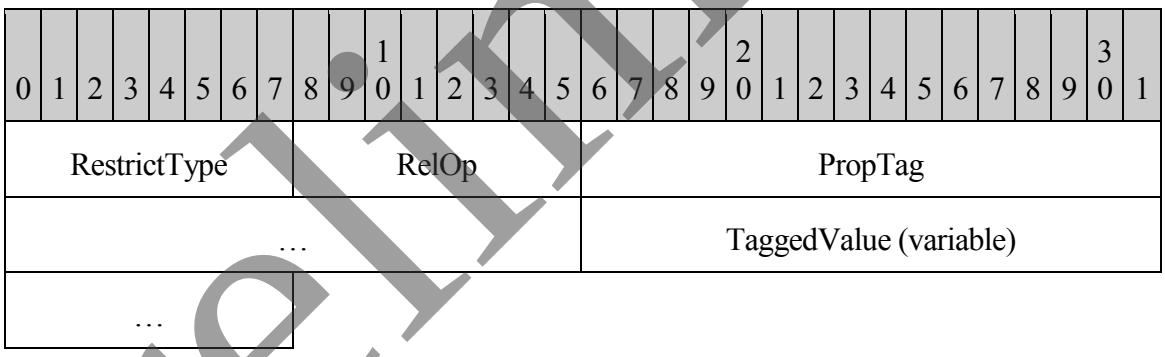
TaggedValue (Variable): A TaggedPropertyValue structure, as specified in section 2.13.4. This structure contains the value to be matched.

The property id portion of the PropertyTag field in TaggedValue is ignored. Its property type MUST match the property type of PropTag.

The result of a content restriction imposed against a property is undefined when the property does not exist. When a client requires well-defined behavior for such a restriction and is not sure whether the property exists, the client SHOULD create an AndRestriction to join the ContentRestriction with an ExistRestriction.

2.14.5 PropertyRestriction

The **PropertyRestriction** structure describes a property restriction which is used to match a constant with the value of a property.



RestrictType (1 byte): Unsigned 8-bit integer. This value indicates the type of restriction and MUST be set to 0x4.

RelOp (1 byte): Unsigned 8-bit integer. The value indicates the relational operator that SHOULD be used to compare the property on the object with **PropValue**. The value MUST be one the following:

Relational operator	Hexadecimal value	Evaluation

	(alternate name)	
LessThan	0x00 RELOP_LT	TRUE if the value of object's property is less than the given value
LessThanOrEqual	0x01 (RELOP_LE)	TRUE if the value of the object's property is less than or equal to the given value
GreaterThan	0x02 (RELOP_GT)	TRUE if the value of the object's property value is greater than the given value
GreaterThanOrEqual	0x03 (RELOP_GE)	TRUE if the value of the object's property value is greater than or equal to the given value
Equal	0x04 (RELOP_EQ)	TRUE if the object's property value equals the given value
NotEqual	0x05 (RELOP_NE)	TRUE if the object's property value does not equal the given value
MemberOfDL	0x64 (RELOP_MEMBER_OF_DL)	TRUE if the value of the object's property is in the DL membership of the specified property value. The value of the object's property MUST be an entryid of a mail-enabled object in the Address Book. The specified property value MUST be an EntryId of a distribution list object in the Address Book.

PropTag (4 bytes): Unsigned 32 bit integer. This value indicates the property tag of the property that MUST be compared.

TaggedValue (Variable): TaggedValue structure (see section 2.13.4). This structure describes the property value to be compared against.

The TaggedValue field contains a property tag subfield which is distinct from the PropTag field of this structure. Only the property type portion of the TaggedValue's property tag subfield is used; the property id is ignored.

Multi-valued properties (when the bit MV_FLAG is set) are supported for this type of restriction, but the base property types (obtained by masking off the bit MV_FLAG) of both the PropTag field and property tag subfield of TaggedValue subfield MUST be the same in all cases.

The MultivalueInstance bit MUST be set in neither the PropTag field nor the property tag subfield of the TaggedValue.

The following table describes which cases are supported for multi-valued properties.

Table 12 Cases supported by multi-valued properties

PropTag	TaggedValue	Support	Details
Single-valued	Single-valued	All RelOp values are supported	Simple comparison
Single-valued	Multi-valued	Only RELOP_EQ and RELOP_NE are supported	The value of property PropTag is compared with each value of TaggedValue. If there are any matches, RELOP_EQ is satisfied. If no matches, RELOP_NE is satisfied.
Multi-valued and same as MultivalueInstance column of table	Single-valued	All RelOp values are supported	The single instance value of property PropTag on the row is compared with TaggedValue.
Multi-valued and same as MultivalueInstancecolumn of table	Multi-valued	Only RELOP_EQ and RELOP_NE supported	The single instance value of property PropTag on the row is compared with each value of TaggedValue. If any matches, RELOP_EQ is satisfied. If no matches, RELOP_NE is satisfied.

Multi-valued but not same as MultivalueInstance of table	Single-valued	All RelOp values supported	Each value of the property PropTag is compared with TaggedValue. For all RelOp values except RELOP_NE, one successful match means the restriction is satisfied. For RELOP_NE, the restriction is satisfied only when there are no matches
Multi-valued but not same as MultivalueInstance of table	Single-valued	Not supported	

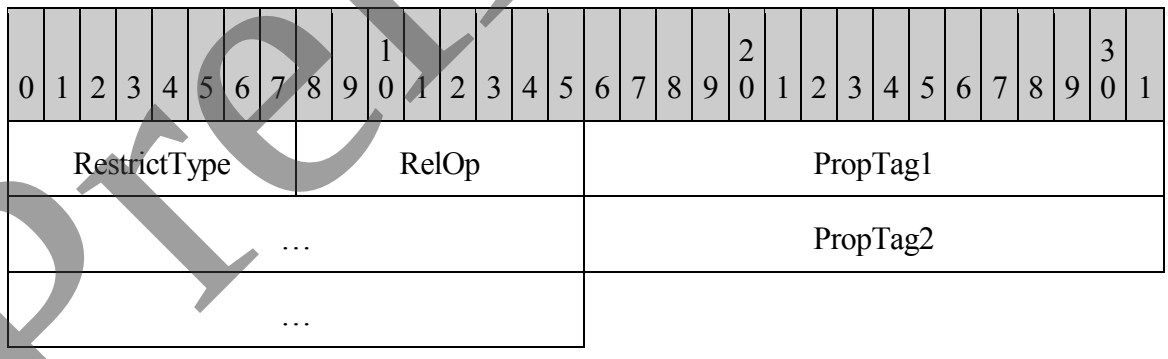
In the context of a RopFindRow or RopRestrict call, the results are undefined if the property PropTag does not exist on the object being tested. By creating an AndRestriction that joins the property restriction with an ExistRestriction, a caller can be guaranteed accurate results.

Only RELOP_EQ and RELOP_NE are allowed for the RelOp field when the base type of PropTag is Boolean.

Note: Some versions of Microsoft MAPI documentation list another relational operator called RELOP_RE. Servers MAY not implement this relational operator, and clients MUST NOT rely on server support for this operator.

2.14.6 ComparePropertiesRestriction

The **ComparePropertiesRestriction** structure specifies a comparison between the values of two properties using a relational operator.



RestrictType (1 byte): Unsigned 8-bit integer. This value indicates the type of restriction and MUST be set to 0x5.

RelOp (1 byte): Unsigned 8-bit integer. The value indicates the relational operator used to compare the two properties. The value MUST be one the following.

Table 13 RelOp Operators Used to Compare Two Properties

Relational operator	Hexadecimal value (alternate name)	Evaluation
LessThan	0x00 RELOP_LT	TRUE if the value of the property specified by Tag1 is less than the value of the property specified by Tag2
LessThanOrEqual	0x01 (RELOP_LE)	TRUE if the value of the property specified by Tag1 is less than or equal to the value of the property specified by Tag2
GreaterThan	0x02 (RELOP_GT)	TRUE if the value of the property specified by Tag1 is greater than the value of the property specified by Tag2
GreaterThanOrEqual	0x03 (RELOP_GE)	TRUE if the value of the property specified by Tag1 is greater than or equal to the value of the property specified by Tag2
Equal	0x04 (RELOP_EQ)	TRUE if the two property values are equal
NotEqual	0x05 (RELOP_NE)	TRUE if the two property values are unequal
MemberOfDL	0x64 (RELOP_MEMBER_OF_DL)	TRUE if the value of the property specified by Tag1 is in the DL membership of Tag2. The value of the property specified by Tag1 MUST be an entryid of a mail-enabled object in Address Book. The value of the property specified by

		Tag2 MUST be an EntryId of a distribution list object in Address Book.

PropTag1 (4 bytes): Unsigned 32 bit integer. This value is the PropertyTag of the first property that MUST be compared.

PropTag2 (4 bytes): Unsigned 32 bit integer. This value is the PropertyTag of the second property that MUST be compared.

The comparison order is *(property tag 1) (relational operator) (property tag 2)*.

The properties to be compared MUST be of the same type.

The result of a compare property value restriction is undefined when one or both of the properties do not exist. When a client requires well-defined behavior for such a restriction and is not sure whether the property exists, for example, it is not a required column of a table, it SHOULD create an AndRestriction to join the compare property restriction with an Exists restriction.

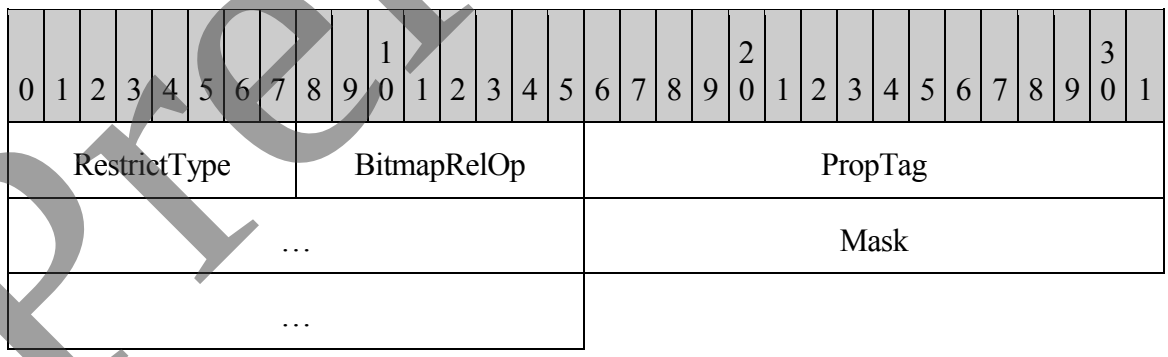
The properties specified by PropTag1 and PropTag2 MUST be single-valued.

Only Equal and NotEqual operators are allowed field when the base type of Tag1 and Tag2 is Boolean.

Note: Some versions of Microsoft MAPI documentation list another relational operator called RELOP_RE. Servers MAY not implement this relational operator, and clients MUST NOT rely on server support for this operator.

2.14.7 BitMaskRestriction

The BitMaskRestriction structure describes a bitmask restriction, which performs a bitwise AND operation and compares the result with zero.



RestrictType (1 byte): Unsigned 8-bit integer. This value indicates the type of restriction and MUST be set to 0x06.

BitmapRelOp (1 byte): Unsigned 8-bit integer. The value specifies how the server MUST perform the masking operation. The value MUST be one of the following:

BMR_EQZ =0x00

Perform a bitwise **AND** operation of the value of Mask with the value of the property PropTag and test for being equal to zero.

BMR_NEZ =0x01

Perform a bitwise **AND** operation of the value of Mask with the value of the property PropTag and test for NOT being equal to zero.

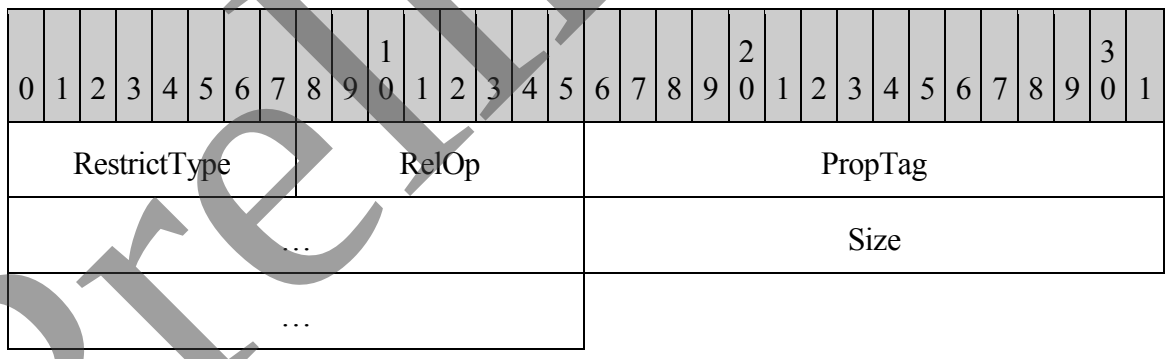
PropTag (4 bytes): Unsigned 32 bit integer. This value is the PropertyTag of the property to be tested. Its property type MUST be single-valued Int32 (refer to section 2.4.2 for details about individual property types).

Mask (4 bytes): Unsigned 32 bit integer. The bitmask to use for the AND operation.

The BitMaskRestriction structure performs a bitwise AND operation using the bitmask Mask and the value of the property PropTag. If the result is zero, then BMR_EQZ is satisfied. If it's nonzero, that is, if the property value has at least one of the same bits set as Mask**,** then BMR_NEZ is satisfied.

2.14.8 SizeRestriction

The SizeRestriction structure describes a size restriction which compares the size (in bytes) of a property value with a given size.



RestrictType (1 byte): Unsigned 8-bit integer. This value indicates the type of restriction and MUST be set to 0x7.

RelOp (1 byte): Unsigned 8-bit integer. The value indicates the relational operator used in the size comparison. The value MUST be one the following:

Table 14 RelOp operators Used for Size Comparison

Relational operator	Hexadecimal value (alternate name)	Evaluation
LessThan	0x00 RELOP_LT	TRUE if the size of object's property is less than the given size
LessThanOrEqual	0x01 (RELOP_LE)	TRUE if the size of object's property is less than or equal to the given size
GreaterThan	0x02 (RELOP_GT)	TRUE if the size of object's property is greater than the given size
GreaterThanOrEqual	0x03 (RELOP_GE)	TRUE if the size of object's property is greater than or equal to the given size
Equal	0x04 (RELOP_EQ)	TRUE if the size of object's property is equal to the given size
NotEqual	0x05 (RELOP_NE)	TRUE if the size of object's property is not equal to the given size

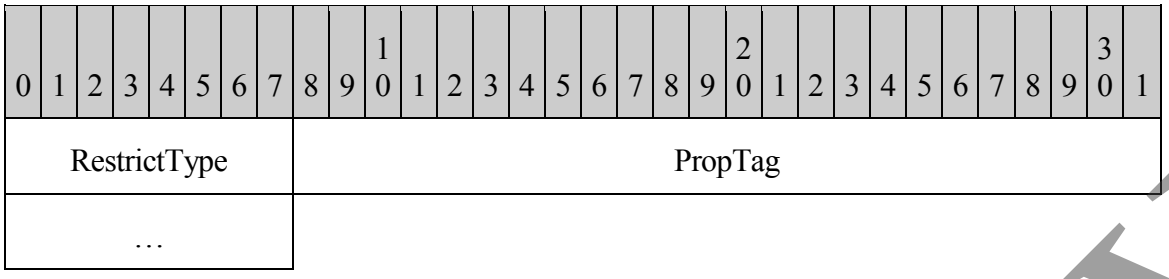
PropTag (4 bytes): Unsigned 32 bit integer. This value indicates the property tag of the property, the size of whose value we are testing.

Size (4 bytes): Unsigned 32 bit integer. This value indicates size, as a count of bytes, that is to be used in the comparison.

In the case where PropTag is multivalued, there are two cases. If it was specified as MULTIVALUEINSTANCE column of the table, the size restriction is evaluated for each row using the size of the single instance value of the row. If was not specified as an MULTIVALUEINSTANCE column of the table, the size restriction is evaluated for each multi-value. If one of the size restrictions succeeds, the restriction is satisfied.

2.14.9 ExistRestriction

The **ExistRestriction** structure tests whether a particular property value exists on a row of the table.



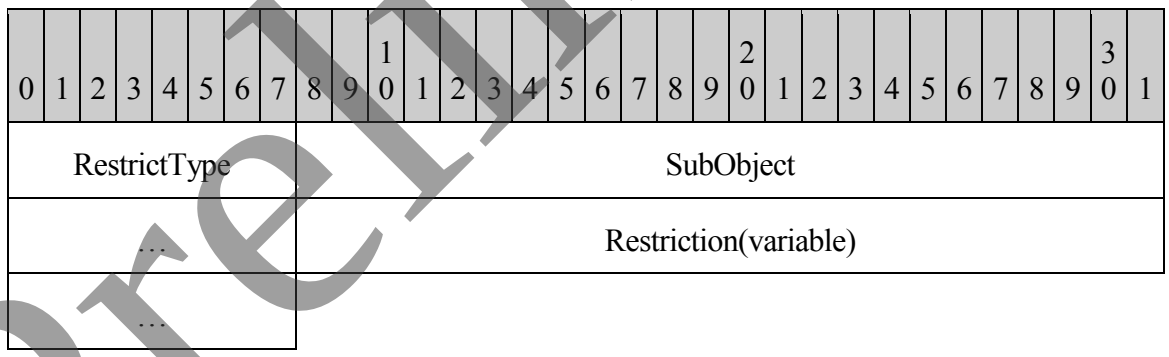
RestrictType (1 byte): Unsigned 8-bit integer. This value indicates the type of restriction and MUST be set to 0x08.

PropTag (4 bytes): Unsigned 32-bit integer. This value is the PropertyTag of the column to be tested for existence in each row.

The ExistRestriction is used to guarantee meaningful results for other types of restrictions that involve properties, such as property and content restrictions. The result of a restriction that involves a property which does not exist on a row is undefined. By creating an AND restriction that joins the property restriction with an ExistRestriction, a client can be guaranteed accurate results.

2.14.10 SubObjectRestriction

The **SubObjectRestriction** structure applies its subrestriction to a message object's attachment table or recipients. If ANY row of the subobject satisfies the subrestriction, then the message satisfies the SubObjectRestriction.



RestrictType (1 byte): Unsigned 8-bit integer. This value indicates the type of restriction and MUST be set to 0x09.

SubObject (4 byte): Unsigned 32-bit integer. This value is a PropertyTag that designates the target of the subrestriction **Restriction.** Only two values are supported:

PidTagMessageRecipients

Apply the subrestriction to a message's recipiens.

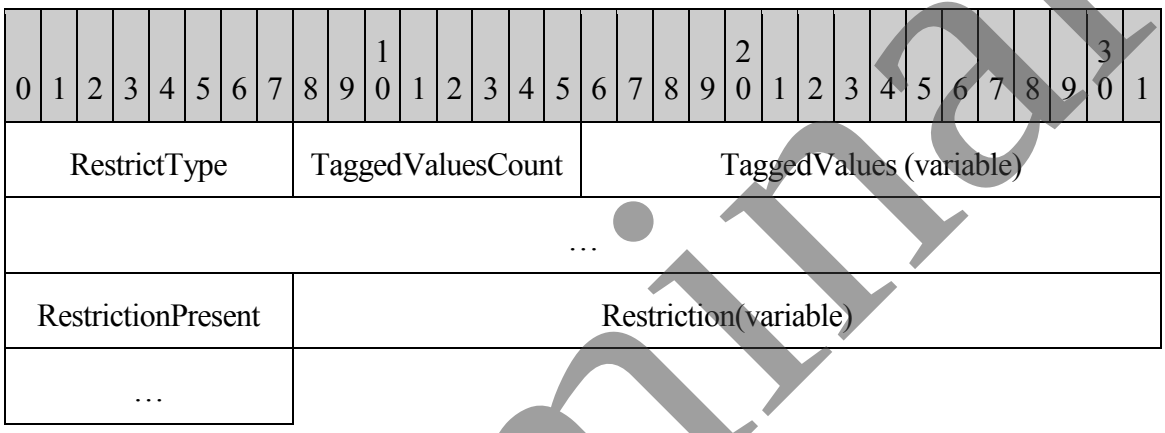
PidTagMessageAttachments

Apply the subrestriction to a message's attachments.

Restriction (variable): A restriction structure. This subrestriction is applied to the rows of the subobject.

2.14.11 CommentRestriction

The **CommentRestriction** structure describes a comment restriction, which is used to annotate a restriction.



RestrictType (1 byte): Unsigned 8-bit integer. This value indicates the type of restriction and MUST be set to 0x0A.

TagggedValuesCount (1 byte): Unsigned 8-bit integer. This value specifies how many TaggedValue structures are present in TaggedValues.

TaggedValues (variable): Array of TaggedPropertyValue (see section 2.13.4) structures. This field MUST contain TaggedValuesCount structures. The TaggedPropertyValue structures MUST NOT include any multi-valued properties.

RestrictionPresent (1 byte): Unsigned 8-bit integer. This field MUST contain either TRUE (0x01) or FALSE (0x00). A TRUE value means that the Restriction field is present, while a FALSE value indicates the Restriction field is not present.

Restriction (variable): A restriction structure. This field is only present if RestrictionPresent is TRUE.

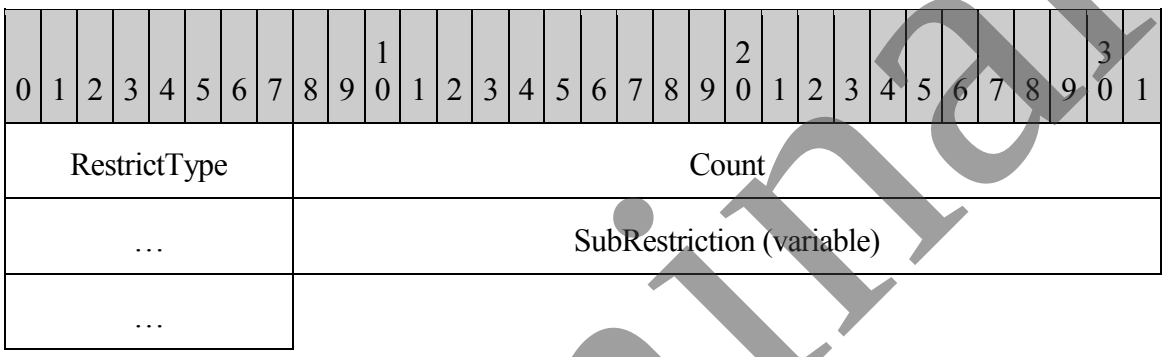
Clients MAY use a CommentRestriction structure to save associated comments together with a restriction they pertain to. The comments are formatted as an arbitrary array of TaggedPropValue structures, and servers MUST store and retrieve this information for the client. If the Restriction field is present, servers MUST evaluate it; if it is not present, then the

CommentRestriction node will effectively evaluate as TRUE. In either case, the comments themselves have no effect on the evaluation of the restriction.

One scenario for using a CommentRestriction is when a restriction includes named properties. A client could store the property names corresponding to the PropertyIds in the comment area, so that if the restriction were later applied to a folder with a different mapping of names to PropertyIds, the names could be remapped to new PropertyIds that are valid for that folder.

2.14.12 CountRestriction

A CountRestriction structure limits the number of matches that SHOULD be returned from its subrestriction.



RestrictType (1 byte): Unsigned 8-bit integer. This value indicates the type of restriction and MUST be set to 0x0B.

Count (4 bytes): Unsigned 32-bit integer. This value specifies the limit on the number of matches to be returned when SubRestriction is evaluated.

SubRestriction (variable): A restriction structure. This field specifies the restriction to be limited.

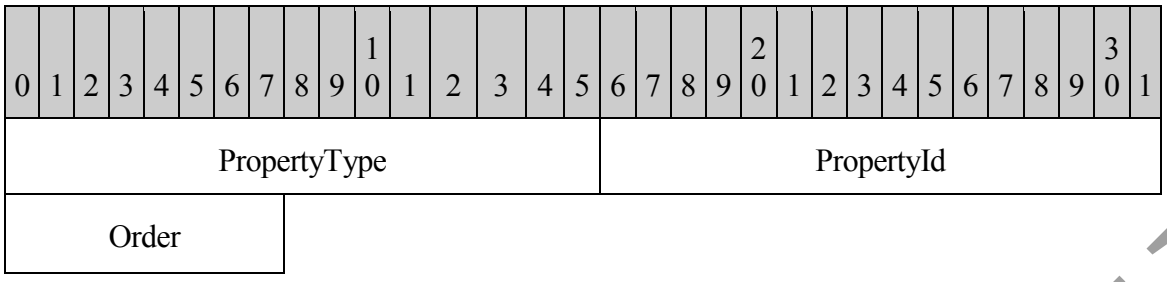
2.15 Sorting

Table sorting is performed by sending a RopSortTable operation to the server. The sort key is specified using a SortOrderSet structure. The SortOrder structure is part of a SortOrderSet. The format of these two structures is specified in the subsections which follow.

2.15.1 SortOrder

The **SortOrder** structure describes one column that is part of a sort key for sorting rows of a table. It gives both the column and the direction of the sort.

SortOrder structures are typically combined into a **SortOrderSet** structure to describe multiple sort keys and directions in a RopSortTable request.



PropertyType (bits 0-15): Identifies the data type of the column to sort on. If the property is multi-valued, for example, the MultivalueFlag bit (0x1000) is set in the PropertyType, then clients MUST also set the MultivalueInstance bit, value 0x2000. In this case the server MUST generate one row for each individual value of a multivalued column, and sort the table by individual values of that column.

PropertyId (bits 16-31): Identifies the column to sort on.

Order (1 byte): MUST be one of the following values:

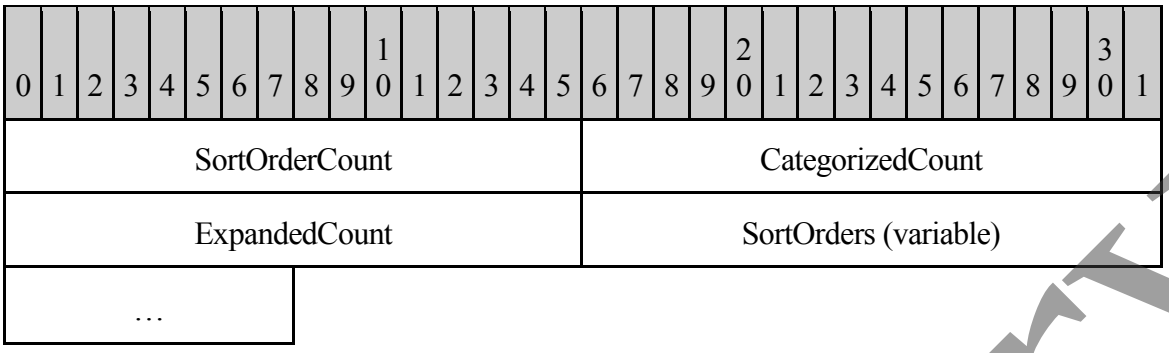
Order name	Order value	Description
Ascending	0x00	Sort by this column in ascending order.
Descending	0x01	Sort by this column in descending order.
MaximumCategory	0x04	Indicates this is an aggregated column in a categorized sort, whose **maximum** value (within the group of items with the same value of the previous category) is to be used as the sort key for the entire group.

If the MultivalueFlag bit is set, then the MultivalueInstance bit MUST also be set, and if the MultivalueInstance bit is set, then the MultivalueFlag bit MUST also be set. In other words, it is not possible to sort on all values of a multivalued column; one row per value MUST be generated and individual values used in the sort.

The MaximumCategory bit causes groups of messages in a categorized sort to be ordered by the maximum value of a column across an entire group. For example, a conversation view is grouped by PidTagConversationTopic; groups are sorted by the group's most recent (maximum) PidTagMessageDeliveryTime value, and within each group messages are sorted by PidTagConversationIndex.

2.15.2 SortOrderSet

The **SortOrderSet** structure describes a sort key consisting of one or more columns.



SortOrderCount (2 bytes): Unsigned 16-bit integer. This value specifies how many SortOrder structures are present in SortOrders.

CategorizedCount (2 bytes): Unsigned 16-bit integer. This value specifies that the first CategorizedCount columns SHOULD be categorized. This value MUST be in the range 0 to SortOrderCount.

ExpandedCount (2 bytes): Unsigned 16-bit integer. This value specifies that the first ExpandedCount of the categorized columns SHOULD start in an expanded state, where all of the rows that apply to the category are visible in the table view. This value MUST be in the range 0 to CategorizedCount.

SortOrders (variable): Array of SortOrder structures. This field MUST contain SortOrderCount structures. At most one of the structures can specify a multi-valued property.

3 Structure Examples

This section provides two examples of how some of these structures would appear as a stream of bytes.

3.1 Restriction Example

The following restriction, described in high level terms, could be used to search for items with reminders set on them.

A restriction of the type RES_AND with the following two sub-clauses:

1. A restriction of type RES_AND, with the following eight sub-clauses:
   - A restriction of type RES_PROPERTY with a relop value of RELOP_NE, comparing the value of PidTagParentEntryId property with the PidTagEntryId of the Deleted Items special folder (see [MS-OXOSFLD])
   1. A restriction of type RES_PROPERTY with a relop value of RELOP_NE, comparing the value of PidTagParentEntryId property

with the PidTagEntryId of the Junk Mail special folder (see [MS-OXOSFLD])

2. A restriction of type RES_PROPERTY with a relop value of RELOP_NE, comparing the value of PidTagParentEntryId property with the PidTagEntryId of the Drafts special folder (see [MS-OXOSFLD])
3. A restriction of type RES_PROPERTY with a relop value of RELOP_NE, comparing the value of PidTagParentEntryId property with the PidTagEntryId of the Outbox special folder (see [MS-OXOSFLD])
4. A restriction of type RES_PROPERTY with a relop value of RELOP_NE, comparing the value of PidTagParentEntryId property with the PidTagEntryId of the Conflicts special folder (see [MS-OXOSFLD])
5. A restriction of type RES_PROPERTY with a relop value of RELOP_NE, comparing the value of PidTagParentEntryId property with the PidTagEntryId of the Local Failures special folder (see [MS-OXOSFLD])
6. A restriction of type RES_PROPERTY with a relop value of RELOP_NE, comparing the value of PidTagParentEntryId property with the PidTagEntryId of the Server Failures special folder (see [MS-OXOSFLD])
7. A restriction of type RES_PROPERTY with a relop value of RELOP_NE, comparing the value of PidTagParentEntryId property with the PidTagEntryId of the Sync Issues special folder (see [MS-OXOSFLD])

2. A restriction of type RES_AND, with the following three sub-clauses:
    1. A restriction of type RES_NOT, with the following sub-clause:
        1. A Restriction of type RES_AND, with the following two sub-clauses:
            1. A Restriction of type RES_EXISTS that specifies the PidTagMessageClass property.
            2. A Restriction of type RES_CONTENT with a FuzzyLevel of FL_PREFIX, comparing the value of PidTagMessageClass property to the string value “IPM.Schedule”

2. A restriction of type RES_BITMASK with a BitmapRelOp value of BMR_EQZ that compares the value of the PidTagMessageFlags property to the ULONG value MSGFLAG_SUBMIT
3. A restriction of type RES_OR, with the following two sub-clauses:
    1. A restriction of type RES_PROPERTY with relop RELOP_EQ, comparing the value of PidLidReminderSet property to the Boolean value 1
    2. A restriction of type RES_AND, with the following two sub-clauses:
        1. A Restriction of type RES_EXISTS that specifies the PidLidRecurring property.
        2. A Restriction of type RES_PROPERTY with relop RELOP_EQ, comparing the value of PidLidRecurring property to the Boolean value 1.

The following describes how this corresponds to a byte stream that is passed between the client and server.

Before formatting this data structure to send to the server, the client would need to send a RopGetPropertyIdsFromNames request to the server to map the two named properties PidLidReminderSet and PidLidRecurring to actual property ids.

Bytes	Field	Meaning
00	RestrictType	RES_AND
02 00	RestrictCount	2
00	RestrictType	RES_AND
08 00	RestrictCount	8
04	RestrictType	RES_PROPERTY
05	RelOp	RELOP_NE
20 10 09 0E	PropTag	PidTagParentEntryId
0E 02	Byte Count	46
00 00 00 00	EntryId Flags	MUST be zero

EE C1 BD 78 61 11 D0 11 91 7B 00 00 00 00 00 01	EntryId Provider UID	UID for Mailbox store
01 00	EntryId Folder Type	eitLTPrivateFolder
(16 byte guid specific to database)	EntryId Message Database GUID	UID identifies database where folder was originally created
(6 bytes identifying Deleted Items folder)	EntryId Global Counter	UID identifies specific folder within database
00 00	EntryId Pad	MUST be zero
04	RestrictType	RES_PROPERTY
05	RelOp	RELOP_NE
20 10 09 0E	PropTag	PidTagParentEntryId
0E 02	Byte Count	46
00 00 00 00	EntryId Flags	MUST be zero
EE C1 BD 78 61 11 D0 11 91 7B 00 00 00 00 00 01	EntryId Provider UID	UID for Mailbox store
01 00	EntryId Folder Type	eitLTPrivateFolder
(16 byte guid specific to database)	EntryId Message Database GUID	UID identifies database where folder was originally created
(6 bytes identifying Junk Mail folder)	EntryId Global Counter	UID identifies specific folder within database
00 00	EntryId Pad	MUST be zero
04	RestrictType	RES_PROPERTY
05	RelOp	RELOP_NE
20 10 09 0E	PropTag	PidTagParentEntryId

0E 02	Byte Count	46
00 00 00 00	EntryId Flags	MUST be zero
EE C1 BD 78 61 11 D0 11 91 7B 00 00 00 00 00 01	EntryId Provider UID	UID for Mailbox store
01 00	EntryId Folder Type	eitLTPrivateFolder
(16 byte guid specific to database)	EntryId Message Database GUID	UID identifies database where folder was originally created
(6 bytes identifying Drafts folder)	EntryId Global Counter	UID identifies specific folder within database
00 00	EntryId Pad	MUST be zero
04	RestrictType	RES_PROPERTY
05	RelOp	RELOP_NE
20 10 09 0E	PropTag	PidTagParentEntryId
0E 02	Byte Count	46
00 00 00 00	EntryId Flags	MUST be zero
EE C1 BD 78 61 11 D0 11 91 7B 00 00 00 00 00 01	EntryId Provider UID	UID for Mailbox store
01 00	EntryId Folder Type	eitLTPrivateFolder
(16 byte guid specific to database)	EntryId Message Database GUID	UID identifies database where folder was originally created
(6 bytes identifying Outbox folder)	EntryId Global Counter	UID identifies specific folder within database
00 00	EntryId Pad	MUST be zero
04	RestrictType	RES_PROPERTY

05	RelOp	RELOP_NE
20 10 09 0E	PropTag	PidTagParentEntryId
0E 02	Byte Count	46
00 00 00 00	EntryId Flags	MUST be zero
EE C1 BD 78 61 11 D0 11 91 7B 00 00 00 00 00 01	EntryId Provider UID	UID for Mailbox store
01 00	EntryId Folder Type	eitLTPrivateFolder
(16 byte guid specific to database)	EntryId Message Database GUID	UID identifies database where folder was originally created
(6 bytes identifying Conflicts folder)	EntryId Global Counter	UID identifies specific folder within database
00 00	EntryId Pad	MUST be zero
04	RestrictType	RES_PROPERTY
05	RelOp	RELOP_NE
20 10 09 0E	PropTag	PidTagParentEntryId
0E 02	Byte Count	46
00 00 00 00	EntryId Flags	MUST be zero
EE C1 BD 78 61 11 D0 11 91 7B 00 00 00 00 00 01	EntryId Provider UID	UID for Mailbox store
01 00	EntryId Folder Type	eitLTPrivateFolder
(16 byte guid specific to database)	EntryId Message Database GUID	UID identifies database where folder was originally created
(6 bytes identifying Local Failures folder)	EntryId Global Counter	UID identifies specific folder within database

00 00	EntryId Pad	MUST be zero
04	RestrictType	RES_PROPERTY
05	RelOp	RELOP_NE
20 10 09 0E	PropTag	PidTagParentEntryId
0E 02	Byte Count	46
00 00 00 00	EntryId Flags	MUST be zero
EE C1 BD 78 61 11 D0 11 91 7B 00 00 00 00 00 01	EntryId Provider UID	UID for Mailbox store
01 00	EntryId Folder Type	eitLTPrivateFolder
(16 byte guid specific to database)	EntryId Message Database GUID	UID identifies database where folder was originally created
(6 bytes identifying Server Failures folder)	EntryId Global Counter	UID identifies specific folder within database
00 00	EntryId Pad	MUST be zero
04	RestrictType	RES_PROPERTY
05	RelOp	RELOP_NE
20 10 09 0E	PropTag	PidTagParentEntryId
0E 02	Byte Count	46
00 00 00 00	EntryId Flags	MUST be zero
EE C1 BD 78 61 11 D0 11 91 7B 00 00 00 00 00 01	EntryId Provider UID	UID for Mailbox store
01 00	EntryId Folder Type	eitLTPrivateFolder
(16 byte guid specific to database)	EntryId Message Database GUID	UID identifies database where folder was

		originally created
(6 bytes identifying Sync Issues folder)	EntryId Global Counter	UID identifies specific folder within database
00 00	EntryId Pad	MUST be zero
00	RestrictType	RES_AND
03 00	RestrictCount	3
02	RestrictType	RES_NOT
00	RestrictType	RES_AND
02 00	RestrictCount	2
08	RestrictType	RES_EXIST
1F 00 1A 00	PropTag	PidTagMessageClass
03	RestrictType	RES_CONTENT
02 00	FuzzyLevelLow	FL_PREFIX
00 00	FuzzyLevelHigh	
1F 00 1A 00	PropertyTag	PidTagMessageClass
49 00 50 00 4D 00 2E 00 53 00 63 00 68 00 65 00 64 00 75 00 6C 00 65 00 00 00	PropValue	“IPM.Schedule”
06	RestrictType	RES_BITMASK
00	BitmapRelOp	BMR_EQZ
03 00 07 0E	PropTag	PidTagMessageFlags
04 00 00 00	Mask	MSGFLAG_SUBMIT
01	RestrictType	RES_OR
02 00	RestrictCount	2

04	RestrictType	RES_PROPERTY
04	RelOp	RELOP_EQ
0B 00 + (2 byte mapped prop id)	PropTag	PidLidReminderSet
01	PropValue	TRUE
00	RestrictType	RES_AND
02 00	RestrictCount	2
08	RestrictType	RES_EXIST
0B 00 + (2 byte mapped prop id)	PropTag	PidLidRecurring
04	RestrictType	RES_PROPERTY
04	RelOp	RELOP_EQ
0B 00 + (2 byte mapped prop id)	PropTag	PidLidRecurring
01	PropValue	TRUE

3.2 PropertyRow Example

In this example, the client sends **RopGetPropsSpecific** to the server requesting the properties from an open message object:

Hexadecimal Value	Property Id	Property Type
0E070003	PidTagMessageFlags	PtypInteger32
00370001	PidTagSubject	PtypUnspecified
1000001F	PidTagBody	PtypString

Additional assumptions used in this example:

- This message had been sent to this mailbox from a different user.
- The message contained an attachment.

- The message had been already read by the user but had not been modified.
- The subject of this message is "Hello".
- The body of the message is so large that the server requires the client to stream the body to the client.

Under these conditions, the PropertyRow data returned from the server would use the FlaggedPropertyRow structure variant (see section 2.13.5) to return the data from **RopGetPropsSpecific** with the following data:

Bytes	Field	Meaning
01	Flag for PropertyRow	There were either errors retrieving values or some values were not returned
00	Flag for FlaggedPropertyValue (see section 2.13.5)	The value for this property is returned
13 00 00 00	Integer32 PropertyValue	MSGFLAG_READ |MSGFLAG_UMODIFIED |MSGFLAG_HASATTACH
1F 00	PropertyType for FlaggedPropertyValueWithType (see 0)	PtypString
00	Flag for FlaggedPropertyValueWithType	RES_PROPERTY
48 00 65 00 6C 00 6C 00 6F 00 00 00	String PropertyValue	"`Hello`"
0A	Flag for FlaggedPropertyValue	The value for this property was not returned. RopOpenStream can be used to obtain the property value.
0E 00 07 80	32-bit SCODE	NotEnoughMemory error (see section 2.4)

4 Security Considerations

There are no special security considerations for this protocol over and above those specified in [MS-OXCRPC].

5 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 that Office/Exchange does not follow the prescription.

<1> Section 2.2.4.3: Outlook 2003 SP3 and Outlook 2007 SP1 sometime leave 3 extra bytes not filled at the end of the Contact Address EntryId structure; in other words, the sum of all fields specified in this protocol can be 3 bytes less than the count of bytes of the entire EntryId. The value extra 3 bytes has no meaning to either the server or the client.

<2> Section 2.2.4.4: Outlook 2003 SP3 and Outlook 2007 SP1 sometimes leaves 3 extra bytes not filled at the end of the Personal Distribution List EntryId structure; in other words, the sum of all fields specified in this protocol can be 3 bytes less than the count of bytes of the entire EntryId. The value extra 3 bytes has no meaning to either the server or the client.

6 Index