[MS-OXCFXICS]:

Bulk Data Transfer Protocol

Intellectual Property Rights Notice for Open Specifications Documentation

- **Technical Documentation.** Microsoft publishes Open Specifications documentation for protocols, file formats, languages, standards as well as overviews of the interaction among each of these technologies.
- **Copyrights**. This documentation is covered by Microsoft copyrights. Regardless of any other terms that are contained in the terms of use for the Microsoft website that hosts this documentation, you may make copies of it in order to develop implementations of the technologies described in the Open Specifications and may distribute portions of it in your implementations using these technologies or your documentation as necessary to properly document the implementation. You may also distribute in your implementation, with or without modification, any schema, IDL's, or code samples that are included in the documentation. This permission also applies to any documents that are referenced in the Open Specifications.
- **No Trade Secrets**. Microsoft does not claim any trade secret rights in this documentation.
- **Patents**. Microsoft has patents that may cover your implementations of the technologies described in the Open Specifications. Neither this notice nor Microsoft's delivery of the documentation grants any licenses under those or any other Microsoft patents. However, a given Open Specification may be covered by Microsoft Open Specification Promise or the Community Promise. If you would prefer a written license, or if the technologies described in the Open Specifications are not covered by the Open Specifications Promise or Community Promise, as applicable, patent licenses are available by contacting iplg@microsoft.com.
- **Trademarks**. The names of companies and products contained in this documentation may be covered by trademarks or similar intellectual property rights. This notice does not grant any licenses under those rights. For a list of Microsoft trademarks, visit www.microsoft.com/trademarks.
- **Fictitious Names**. The example companies, organizations, products, domain names, e-mail addresses, logos, people, places, and events depicted in this documentation are fictitious. No association with any real company, organization, product, domain name, email address, logo, person, place, or event is intended or should be inferred.

Reservation of Rights. All other rights are reserved, and this notice does not grant any rights other than specifically described above, whether by implication, estoppel, or otherwise.

Tools. The Open Specifications do not require the use of Microsoft programming tools or programming environments in order for you to develop an implementation. If you have access to Microsoft programming tools and environments you are free to take advantage of them. Certain Open Specifications are intended for use in conjunction with publicly available standard specifications and network programming art, and assumes that the reader either is familiar with the aforementioned material or has immediate access to it.

Revision Summary

Date	Revision History	Revision Class	Comments
4/4/2008	0.1		Initial Availability.
4/25/2008	0.2		Revised and updated property names and other technical content.
6/27/2008	1.0		Initial Release.
8/6/2008	1.01		Revised and edited technical content.
9/3/2008	1.02		Revised and edited technical content.
12/3/2008	1.03		Revised and edited technical content.
2/4/2009	1.04		Revised and edited technical content.
3/4/2009	1.05		Editorial updates.
4/10/2009	2.0		Updated technical content and applicable product releases.
7/15/2009	3.0	Major	Revised and edited for technical content.
11/4/2009	4.0.0	Major	Updated and revised the technical content.
2/10/2010	5.0.0	Major	Updated and revised the technical content.
5/5/2010	6.0.0	Major	Updated and revised the technical content.
8/4/2010	6.1	Minor	Clarified the meaning of the technical content.
11/3/2010	7.0	Major	Significantly changed the technical content.
3/18/2011	8.0	Major	Significantly changed the technical content.
8/5/2011	9.0	Major	Significantly changed the technical content.
10/7/2011	10.0	Major	Significantly changed the technical content.
1/20/2012	11.0	Major	Significantly changed the technical content.
4/27/2012	12.0	Major	Significantly changed the technical content.
7/16/2012	12.0	No Change	No changes to the meaning, language, or formatting of the technical content.
10/8/2012	13.0	Major	Significantly changed the technical content.
2/11/2013	14.0	Major	Significantly changed the technical content.
7/26/2013	15.0	Major	Significantly changed the technical content.
11/18/2013	16.0	Major	Significantly changed the technical content.
2/10/2014	16.0	No Change	No changes to the meaning, language, or formatting of the technical content.
4/30/2014	16.1	Minor	Clarified the meaning of the technical content.
7/31/2014	16.2	Minor	Clarified the meaning of the technical content.
10/30/2014	16.2	No Change	No changes to the meaning, language, or formatting of the

Date	Revision History	Revision Class	Comments
			technical content.
3/16/2015	17.0	Major	Significantly changed the technical content.
5/26/2015	18.0	Major	Significantly changed the technical content.
9/14/2015	18.1	Minor	Clarified the meaning of the technical content.

Table of Contents

1 Introduction

The Bulk Data Transfer Protocol enables the bulk transmission of **mailbox** data, represented by folders and messages, between clients and servers. This protocol is commonly used for replicating, exporting, or importing mailbox content between clients and servers.

This protocol describes the following:

- How a client can configure a **remote operation (ROP)** to upload a set of folders or messages to a server, or download a set of folders or messages from a server.
- How a client or a server can receive and reconstitute folders and messages that are transmitted from another client or another server.
- How a client can upload changes made to local folders and message replicas to a server.
- Semantics of ROPs that are used to fulfill the aforementioned operations.

Sections 1.8, 2, and 3 of this specification are normative and can contain the terms MAY, SHOULD, MUST, MUST NOT, and SHOULD NOT as defined in [RFC2119]. Sections 1.5 and 1.9 are also normative but do not contain those terms. All other sections and examples in this specification are informative.

1.1 Glossary

The following terms are specific to this document:

ASCII: The American Standard Code for Information Interchange (ASCII) is an 8-bit character-encoding scheme based on the English alphabet. ASCII codes represent text in computers, communications equipment, and other devices that work with text. ASCII refers to a single 8-bit ASCII character or an array of 8-bit ASCII characters with the high bit of each character set to zero.

Attachment object: A set of properties that represents a file, **Message object**, or structured storage that is attached to a Message object and is visible through the attachments table for a Message object.

Augmented Backus-Naur Form (ABNF): A modified version of Backus-Naur Form (BNF), commonly used by Internet specifications. ABNF notation balances compactness and simplicity with reasonable representational power. ABNF differs from standard BNF in its definitions and uses of naming rules, repetition, alternatives, order-independence, and value ranges. For more information, see [RFC5234].

base property type: The type of the property, if the property is single-valued, or the type of an element of the property, if the property is multi-valued.

best body: The text format that provides the richest representation of a **message body**. The algorithm for determining the best-body format is described in [MS-OXBBODY].

camel-cased: The capitalization style applied to compound words or phrases when they are joined without spaces and the first letter of each word, except the first word, is capitalized within the compound. For example, displayName is camel-cased.

change number: A number that identifies a version of a messaging object. A change number is identical in format to a message ID (MID) or folder ID (FID).

checkpoint ICS state: An **Incremental Change Synchronization (ICS)** state that is provided by a server in the middle of an ICS operation, which reflects the state of the **local replica**,

indicated by the **initial ICS state**, after applying all differences transmitted in the ICS operation.

code page: An ordered set of characters of a specific script in which a numerical index (code-point value) is associated with each character. Code pages are a means of providing support for character sets (1) and keyboard layouts used in different countries. Devices such as the display and keyboard can be configured to use a specific code page and to switch from one code page (such as the United States) to another (such as Portugal) at the user's request.

common byte stack: A list of arrays of bytes. Byte values of contained arrays, when together in their natural order, represent common high-order bytes of GLOBCNT values. Common byte stacks are used in a last-in first-out (LIFO) fashion during serialization or deserialization of GLOBSETs.

conflict detection: A process that is used to determine whether two versions of the same object conflict with each other, that is, one is not a direct or indirect predecessor of another.

conflict handling: One or more actions that are taken upon detection of a conflict between versions of the same object. These actions include conflict reporting and conflict resolution.

conflict reporting: An automated process that notifies a system actor of a previously detected conflict.

conflict resolution: An automated or semi-automated process that is used to resolve a previously detected conflict between versions of an object. The process replaces conflicting versions with a successor version. How a successor version relates to a conflicting version depends on the algorithm that is used.

content synchronization: The process of keeping synchronized versions of **Message objects** and their properties on a client and server.

deleted item list: An abstract repository of information about deleted items.

Deleted Items folder: A special folder that is the default location for objects that have been deleted.

Embedded Message object: A **Message object** that is stored as an **Attachment object** within another Message object.

enterprise/site/server distinguished name (ESSDN): An X500 DN that identifies an entry in an abstract naming scheme that is separate from an address book. The naming scheme defines enterprises, which contain sites, and sites contain servers and users. There is no concrete data structure that embodies an ESSDN. Instead, an address book entry can contain an ESSDN as a property of the entry.

EntryID: A sequence of bytes that is used to identify and access an object.

expired Message object: A **Message object** that was removed by a server due to the age of the Message object.

external identifier: A globally unique identifier for an entity that represents either a **foreign identifier** or an **internal identifier**. It consists of a GUID that represents a namespace followed by one or more bytes that contain an identifier for an entity within that namespace. If an external identifier represents an internal identifier, it can be also called a **global identifier**.

FastTransfer context: Either a **FastTransfer download context** or a **FastTransfer upload context**.

FastTransfer download context: A Server object that represents a context for a FastTransfer download.

FastTransfer stream: A binary format for encoding full or partial folder and message data. It can also encode information about differences between mailbox replicas.

FastTransfer upload context: A Server object that represents a context for a FastTransfer upload.

final ICS state: An **Incremental Change Synchronization (ICS)** state that is provided by a server upon completion of an ICS operation. A final ICS state is a **checkpoint ICS state** that is provided at the end of the ICS operation.

flags: A set of values used to configure or report options or settings.

folder associated information (FAI): A collection of **Message objects** that are stored in a Folder object and are typically hidden from view by email applications. An FAI Message object is used to store a variety of settings and auxiliary data, including forms, views, calendar options, favorites, and category lists.

Folder object: A messaging construct that is typically used to organize data into a hierarchy of objects containing Message objects and **folder associated information (FAI)** Message objects.

foreign identifier: An identifier that is assigned to an entity by a foreign system, typically a client. It always has a form of an **external identifier**, but not all external identifiers are foreign identifiers.

global identifier: A form of encoding for an internal identifier that makes it unique across all stores. Global identifiers are a subset of **external identifiers**, and they consist of a **REPLGUID** followed by a 6-byte global counter.

globally unique identifier (GUID): A term used interchangeably with universally unique identifier (UUID) in Microsoft protocol technical documents (TDs). Interchanging the usage of these terms does not imply or require a specific algorithm or mechanism to generate the value. Specifically, the use of this term does not imply or require that the algorithms described in [RFC4122] or [C706] must be used for generating the **GUID**. See also universally unique identifier (UUID).

handle: Any token that can be used to identify and access an object such as a device, file, or a window.

hard delete: A process that removes an item permanently from the system. If an item is hard deleted, a server does not retain a back-up copy of the item and a client cannot access or restore the item. See also **soft delete**.

hierarchy synchronization: The process of keeping synchronized versions of folder hierarchies and their properties on a client and server.

Incremental Change Synchronization (ICS): A data format and algorithm that is used to synchronize folders and messages between two sources.

initial ICS state: An **Incremental Change Synchronization (ICS)** state that is provided by a client when it configures an ICS operation.

internal identifier: A Folder ID or Message ID, as described in [MS-OXCDATA].

interpersonal messaging subtree: The root of the hierarchy of folders commonly visible in a messaging client. This includes mailbox folders (such as the Inbox folder and Outbox folder) and user-created folders, including user-created public folders.

little-endian: Multiple-byte values that are byte-ordered with the least significant byte stored in the memory location with the lowest address.

local replica: A copy of the data in a **mailbox** that exists on the client.

mailbox: A **message store** that contains email, calendar items, and other **Message objects** for a single recipient.

marker: An unsigned 32-bit integer value that adheres to **property tag** syntax and is used to denote the start and end of related data in a **FastTransfer stream**. The property tags that are used by markers do not represent valid properties.

message body: The main message text of an email message. A few properties of a **Message object** represent its message body, with one property containing the text itself and others defining its **code page** and its relationship to alternative body formats.

Message object: A set of properties that represents an email message, appointment, contact, or other type of personal-information-management object. In addition to its own properties, a Message object contains recipient properties that represent the addressees to which it is addressed, and an attachments table that represents any files and other Message objects that are attached to it.

message store: A unit of containment for a single hierarchy of Folder objects, such as a mailbox or public folders.

messaging object: An object that exists in a **mailbox**. It can be only a **Folder object** or a **Message object**.

meta-property: An entity that is identified with a **property tag** containing information (a value) that describes how to process other data in a **FastTransfer stream**.

named property: A property that is identified by both a GUID and either a string name or a 32-bit identifier.

normal message: A message that is not a **folder associated information (FAI)** message.

offline: The condition of not being connected to or not being on a network or the Internet. Offline can also refer to a device, such as a printer that is not connected to a computer, and files that are stored on a computer that is not connected to or not on a network or the Internet.

Outbox folder: A special folder that contains **Message objects** that are submitted to be sent.

partial completion: The outcome of a complex operation with independent steps, where some steps succeeded and some steps failed.

Pascal-cased: The capitalization style applied to compound words or phrases when they are joined without spaces and the first letter of each word (including the first word) is capitalized within the compound. For example, DisplayName is a pascal-cased.

Predecessor Change List (PCL): A set of **change numbers** that specify the latest versions of a messaging object in all replicas that were integrated into the current version. It is used for conflict detection.

property ID: A 16-bit numeric identifier of a specific attribute (1). A property ID does not include any **property type** information.

property list restriction table: A set of restrictions, expressed in tabular form, that is imposed on an array of properties and the values of those properties.

property set: A set of attributes (1), identified by a **GUID**. Granting access to a property set grants access to all the attributes in the set.

property tag: A 32-bit value that contains a property type and a property ID. The low-order 16 bits represent the property type. The high-order 16 bits represent the property ID.

property type: A 16-bit quantity that specifies the data type of a property value.

public folder: A **Folder object** that is stored in a location that is publicly available.

recipient: An entity that can receive email messages.

Recipient object: A set of properties that represent the recipient of a **Message object**.

remote operation (ROP): An operation that is invoked against a server. Each ROP represents an action, such as delete, send, or query. A ROP is contained in a ROP buffer for transmission over the wire.

remote procedure call (RPC): A context-dependent term commonly overloaded with three meanings. Note that much of the industry literature concerning RPC technologies uses this term interchangeably for any of the three meanings. Following are the three definitions: (*) The runtime environment providing remote procedure call facilities. The preferred usage for this meaning is "RPC runtime". (*) The pattern of request and response message exchange between two parties (typically, a client and a server). The preferred usage for this meaning is "RPC exchange". (*) A single message from an exchange as defined in the previous definition. The preferred usage for this term is "RPC message". For more information about RPC, see [C706].

replica: (1) A server that hosts an instance of a message item in a folder.

(2) A copy of the data that is in a user's **mailbox** at a specific point in time.

replica GUID (REPLGUID): A value that represents a namespace for identifiers. If a REPLGUID is combined with a GLOBSET, the result is a set of **global identifiers**. A REPLGUID value has an associated **replica ID (REPLID)** that is used in its place on disk and on the wire.

replica ID (REPLID): A value that is mapped to a **replica GUID (REPLGUID)** that identifies a namespace for IDs within a given logon. REPLIDs are used on disk and on the wire for compactness, and are replaced with the corresponding REPLGUID for external consumption.

Rich Text Format (RTF): Text with formatting as described in [MSFT-RTF].

ROP request: See **ROP request buffer**.

ROP request buffer: A ROP buffer that a client sends to a server to be processed.

ROP response: See **ROP response buffer**.

ROP response buffer: A ROP buffer that a server sends to a client to be processed.

Sent Items folder: A special folder that is the default location for storing copies of **Message objects** after they are submitted or sent.

server replica: A copy of a user's **mailbox** that exists on a server.

soft delete: A process that removes an item from the system, but not permanently. If an item is soft deleted, a server retains a back-up copy of the item and a client can access, restore, or permanently delete the item. See also **hard delete**.

synchronization context: See **synchronization download context** or **synchronization upload context**.

synchronization download context: A Server object that represents a context for an **ICS** download.

synchronization scope: A set of complex criterion that defines a superset of all the **messaging objects** that are within a specific mailbox and are considered for a single synchronization operation.

synchronization type: The type of synchronization that is occurring, either a **hierarchy synchronization** or a **content synchronization**.

synchronization upload context: A Server object that represents a context for an **ICS** upload.

top-level message: A message that is not included in another message as an **Embedded Message object**. Top-level messages are **messaging objects**.

Unicode: A character encoding standard developed by the Unicode Consortium that represents almost all of the written languages of the world. The **Unicode** standard [UNICODE5.0.0/2007] provides three forms (UTF-8, UTF-16, and UTF-32) and seven schemes (UTF-8, UTF-16, UTF-16 BE, UTF-16 LE, UTF-32, UTF-32 LE, and UTF-32 BE).

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

1.2 References

Links to a document in the Microsoft Open Specifications library point to the correct section in the most recently published version of the referenced document. However, because individual documents in the library are not updated at the same time, the section numbers in the documents may not match. You can confirm the correct section numbering by checking the Errata.

1.2.1 Normative References

We conduct frequent surveys of the normative references to assure their continued availability. If you have any issue with finding a normative reference, please contact dochelp@microsoft.com. We will assist you in finding the relevant information.

[MS-OXBBODY] Microsoft Corporation, "Best Body Retrieval Algorithm".

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

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

[MS-OXCMAPIHTTP] Microsoft Corporation, "Messaging Application Programming Interface (MAPI) Extensions for HTTP".

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

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

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

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

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

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

[MS-OXOMSG] Microsoft Corporation, "Email Object Protocol".

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

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

[RFC5234] Crocker, D., Ed., and Overell, P., "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, January 2008, http://www.rfc-editor.org/rfc/rfc5234.txt

1.2.2 Informative References

[MS-OXCMAIL] Microsoft Corporation, "RFC 2822 and MIME to Email Object Conversion Algorithm".

[MS-OXCSPAM] Microsoft Corporation, "Spam Confidence Level Protocol".

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

[MS-OXOCAL] Microsoft Corporation, "Appointment and Meeting Object Protocol".

[MS-OXOFLAG] Microsoft Corporation, "Informational Flagging Protocol".

[MS-OXORMDR] Microsoft Corporation, "Reminder Settings Protocol".

[MS-OXOTASK] Microsoft Corporation, "Task-Related Objects Protocol".

[MS-OXPROTO] Microsoft Corporation, "Exchange Server Protocols System Overview".

1.3 Overview

This protocol describes how clients and servers can efficiently exchange data that is represented as folders and messages that are contained in private or public mailboxes.

Efficiency in the exchange of data is achieved through the following means:

- Packaging data for several folders or messages into a single ROP response, which can be compressed at the **remote procedure call (RPC)** level.
- Reducing transmitted data to only changes that the user is interested in.
- Reducing transmitted data to only changes that relate to a subset of folder or message data by using **Incremental Change Synchronization (ICS)**.
- Performing optimizations on the server provided that the server knows the scope of the operation ahead of time.
- Minimizing the bandwidth required to copy message and folder content by efficiently packing data by using **FastTransfer streams**.

This protocol supports the transfer of data in scenarios that derive from the following semi-independent variables:

1. Direction of data transmission: download or upload.
2. Type of **messaging objects** included in a transmission: folders, messages, or both.
3. Scope of the data that is transmitted for a messaging object. The scope might be one of the following:
    - A full object or a subset of its data.
    - Changes since the last transmission.
    - Operations such as a read state change or a move.
4. Scope of the messaging objects that are included in a set. The scope might be one of the following:
    - Identified directly by **Folder object** identifiers and **Message object** identifiers.
    - Identified by a combination of criteria and state information maintained by the client.

This specification is based on the following roles: one server, and one or more clients. The only exception is the server-to-client-to-server upload scenario described in section 3.3.4.2.1.

1.3.1 FastTransfer Copy Operations

FastTransfer copy operations enable clients to efficiently copy the content of explicitly specified folders, messages, and attachments between **replicas (1)** of the same or different mailboxes by using a special binary format known as a FastTransfer stream as the medium. A FastTransfer stream contains copies of folder, message, or attachment content in a predefined serialized format, as described in section 2.2.4. The FastTransfer stream can be used to create copies of this folder, message, or attachment content in any destination folder, on any mailbox, on any client, or on any server.

Every FastTransfer operation is independent. After the operation is complete, no state has to be maintained on the client or on the server.

FastTransfer download operations enable clients to download a copy of the explicitly specified folders, messages, or attachments in the FastTransfer stream format. The resulting FastTransfer stream can be either interpreted on the client, or used in a FastTransfer upload operation if the intent is to copy messaging objects between mailboxes on different servers. FastTransfer download operations are described in section 2.2.3.1.1.

FastTransfer upload operations enable a client to create new folders or modify content of existing folders, messages, and attachments by encoding data into the FastTransfer stream format. FastTransfer upload operations are described in section 2.2.3.1.2.

FastTransfer operations are used in the ICS download process to download changes and deletions to mailbox data. FastTransfer operations are not used in the ICS upload process.

1.3.2 Incremental Change Synchronization

ICS enables servers and clients to keep synchronized versions of messages, folders, and their related properties on both systems. Changes that are made to messages and folders on the client are replicated to the server and vice versa. ICS can determine differences between two folder hierarchies or two sets of content, and can upload or download information about the differences in a single session.

Changes to folder properties, changes to the folder hierarchy, and folder creations and deletions are included in **hierarchy synchronization** operations.

Changes to message properties, changes to read and unread message state, changes to **recipients** and attachment information, message creations, and message deletions are included in **content synchronization** operations.

Hierarchy synchronization and content synchronization operations are the actual processes used to implement ICS on the client and server.

1.3.2.1 Download Changes Using ICS

To download changes and deletions made to mailbox data on the server, the client first creates a **synchronization download context**, as described in section 2.2.3.2.1. Once the synchronization download context has been created, the client uploads a set of properties, the ICS state properties, as described in section 2.2.1.1, that track the changes currently on the client. The server then uses the data in the ICS state properties and the **synchronization scope** to determine the set of differences to download to the client.

Once the state information has been uploaded and the server has determined the changes to download, that information is downloaded to the client through one or more iterations of a single Fast

Transfer ROP, as described in section 2.2.3.1.1.5, whose response buffer can be efficiently packed at the RPC level. The FastTransfer ROP is used in ICS download operations only.ICS upload operations use an independent set of ICS ROPs, as described in section 2.2.3.2.4.

Performing a hierarchy synchronization download operation using a synchronization download context that was opened on a folder produces information about all folder changes and folder deletions of descendants of that folder that have happened since the last synchronization download, as defined by the **initial ICS state**.

Performing a content synchronization download operation using a synchronization download context that was opened on a folder produces information about all message changes and message deletions in the folder that have occurred since the last synchronization download. ICS uses the ICS state properties, as described in section 2.2.1.1, to determine the differences, as defined by the initial ICS state.

1.3.2.2 Upload Changes Using ICS

Similar to the ICS process for downloading changes, the process for uploading changes using ICS requires that the client first creates a **synchronization upload context**, as described in section 2.2.3.2.1. Once the synchronization upload context has been created, the client uploads a set of properties, the ICS state properties, as described in section 2.2.1.1, that track the changes currently on the client. The server then accepts the changes from the client and sends the **final ICS state**, containing the updated ICS properties, to the client. The ICS state properties are updated to reflect the items that were uploaded to the server so that server does not download the same changes to the client the next time the ICS process occurs.

The process of uploading the mailbox changes from the client to the server resembles the ICS download process, except that instead of streaming data through a single FastTransfer ROP, multiple individual ICS ROPs are sent to upload changes to individual objects within a mailbox, as described in section 2.2.3.2.4.

This protocol supports the uploading of hierarchy differences, such as creation and deletion of folders and changes to folder properties.

This specification also supports the uploading of differences in the contents of folders, such as creation and deletion of messages, changes to message properties and read state, and the moving of messages between folders.

1.4 Relationship to Other Protocols

This protocol provides a low-level explanation of bulk data transfer operations.

This protocol relies on the following:

- An understanding of ROPs, as described in [MS-OXCROPS].
- An understanding of folders and messages, as described in [MS-OXCFOLD] and [MS-OXCMSG], respectively.

For conceptual background information and overviews of the relationships and interactions between this and other protocols, see [MS-OXPROTO].

1.5 Prerequisites/Preconditions

When performing bulk data transfer operations, this protocol assumes that the client has previously logged on to the server and has acquired a **handle** to the folder that contains the messages and subfolders that will be uploaded or downloaded. For information about folders, see [MS-OXCFOLD].

1.6 Applicability Statement

This protocol was designed for the following uses:

- To support the replication of mailbox content between clients and servers.
- To support client-driven copying of data between multiple mailboxes on multiple servers.
- To support exporting or importing of data to or from a mailbox.

This protocol provides high efficiency and complete preservation of data fidelity for the uses described in this section. However, use of the protocol is not appropriate in the following scenarios:

- For those copying data between folders in the same mailbox, or different mailboxes residing on the same server. Consider using the **RopCopyTo** ROP, as described in [MS-OXCROPS] section 2.2.8.12, for maximum efficiency.
- For those requiring detailed control over the set of information that has to be transferred for each message. Consider using other ROPs described in [MS-OXCROPS] that provide access to individual parts of messages.
- For those that impose constraints on the amount of data that has to be passed over the wire or stored on the client.
- For those that do not allow for persistence of state information on the client between runs.

1.7 Versioning and Capability Negotiation

This document covers versioning issues in the following areas:

- **Localization:** Localization-related aspects of the protocol are described in section 2.2.3.1.1.1.2.
- **Capability Negotiation:** This protocol performs explicit capability negotiation by using the following ROPs, properties, and **flags**. Support of the following features is determined by the versions of the client and server that are supplied either by the **EcDoConnectEx** method, as described in [MS-OXCRPC], or by the **X-ClientApplication** and **X-ServerApplication** headers of the **Connect** request type request and response, as described in [MS-OXCMAPIHTTP]. Both the client and server limit their behavior to the capabilities supported by the other. For more information, see [MS-OXCRPC] section 3.1.4.1.3.

Client version	Description
11.0.0.4920 and above	The client supports receiving the 0x00000480 value in the **ReturnValue** field of the **RopFastTransferSourceGetBuffer** ROP (section 2.2.3.1.1.5) response.
12.0.3730.0 and above	The client supports send optimization for ICS using the **PidTagTargetEntryId** property ([MS-OXOMSG] section 2.2.1.76). For more information, see section 3.3.4.3.3.2.1.2.

Server version	Description
8.0.359.0 and above	The server supports the **PartialItem** flag of the **SendOptions** field, as described in section 2.2.3.2.1.1.1. Earlier server versions do not support this flag.

The **RopTellVersion** ROP is used to explicitly declare capabilities of the servers in the server-to-client-to-server upload scenario. For more information, see section 3.3.4.2.1.

1.8 Vendor-Extensible Fields

This protocol provides no extensibility beyond what is specified in [MS-OXCMSG].

1.9 Standards Assignments

None.

2 Messages

2.1 Transport

The **ROP request buffers** and **ROP response buffers** specified by this protocol are sent to and received by the server by using the underlying Remote Operations (ROP) List and Encoding Protocol, as specified in [MS-OXCROPS].

2.2 Message Syntax

The following notations are used in this specification:

- **MetaTagCnset***. Refers to any of the following properties: **MetaTagCnsetSeen** (section 2.2.1.1.2), **MetaTagCnsetSeenFAI** (section 2.2.1.1.3), and **MetaTagCnsetRead** (section 2.2.1.1.4).
- **RopFastTransferSourceCopy***. Refers to any of the following ROPs: **RopFastTransferSourceCopyTo** (section 2.2.3.1.1.1), **RopFastTransferSourceCopyProperties** (section 2.2.3.1.1.2), **RopFastTransferSourceCopyMessages** (section 2.2.3.1.1.3), and **RopFastTransferSourceCopyFolder** (section 2.2.3.1.1.4).
- **RopSynchronizationImport***. Refers to any of the following ROPs: **RopSynchronizationImportMessageChange** (section 2.2.3.2.4.2), **RopSynchronizationImportHierarchyChange** (section 2.2.3.2.4.3), **RopSynchronizationImportMessageMove** (section 2.2.3.2.4.4), **RopSynchronizationImportDeletes** (section 2.2.3.2.4.5), and **RopSynchronizationImportReadStateChanges** (section 2.2.3.2.4.6).

Section 2.2.3.2.4.2 through section 2.2.4.3.25 use **property list restriction tables** in the following format to describe restrictions on arrays of property values:

Name	Restrictions	Comments
PidSomeProperty	Conditional Fixed position ...	When the value of Restrictions column contains "Conditional", the Comments column is specifying the condition of existence. When the value of the Restrictions column does not contain "Conditional", the Comments column can contain comments about the property value.
< other properties >	*Prohibited*	Comments.

Any property cannot exist in a property list restriction table more than once. All nonitalicized rows of the table represent a restriction that is imposed on the property identified in the Name column. For more information about all possible properties, see [MS-OXPROPS]. The Comments column contains free-form comments that amend the meaning of the Name and Restrictions columns. The Restrictions column specifies a subset of the following restrictions:

- Optional [default]: The property can be present in the array.
- Required: The property MUST be present in the array.
- Fixed position: The position of the property within the array is fixed and MUST correspond to the position of the corresponding restriction in the property list restriction table.
- Conditional: The presence of the property in the array is conditional. See the Comments column for conditions.

- Prohibited: The property MUST NOT be present in the array. Italicized rows represent restrictions that apply to special sets of properties. The special set < *other properties* > represents all properties that are not mentioned in the property list restriction table explicitly.
- No restrictions: There are no restrictions on the property in the array. The property does not have a fixed position in the array, and by default, the property's presence is optional in the array.

All undefined bits in flag structures and undefined values of enumerations that are defined in this specification are reserved; clients MUST pass 0. Server behavior for undefined flags and enumeration values is defined in section 3.2.

2.2.1 Properties

2.2.1.1 ICS State Properties

ICS uses a set of properties known as the ICS state properties to enable a server to narrow down the set of data passed during an Incremental Change Synchronization (ICS) to the client. The ICS state properties specify the state of the **local replica**, bounded by the synchronization scope that is configured by the **RopSynchronizationConfigure** ROP (section 2.2.3.2.1.1). Details about the ROPs used to upload the ICS state properties to the server are included in section 2.2.3.2.2. Details about how the ICS state properties are used by the client and server are included in sections 3.3.5.2 and 3.2.5.2 respectively.

All ICS state properties are of the **PtypBinary** type ([MS-OXCDATA] section 2.11.1), and contain a serialized **IDSET** structure in the **replica GUID (REPLGUID)**-based form, as specified in section 2.2.2.4.2. For details on serializing an **IDSETs**, see section 3.1.5.4. For details on creating a compact **IDSET**, see section 3.1.5.5.

All properties specified in this section are part of the ICS state. Two of these properties are used for hierarchy synchronization operations: **MetaTagIdsetGiven** (section 2.2.1.1.1) and **MetaTagCnsetSeen** (section 2.2.1.1.2). All four properties are used for content synchronization operations.

The ICS state specifies the state of the local replica bounded by the synchronization scope included by the client in the **RopSynchronizationConfigure** ROP request (section 2.2.3.2.1.1).

2.2.1.1.1 MetaTagIdsetGiven ICS State Property

Property ID: 0x4017

Data type: **PtypInteger32**, 0x0003 ([MS-OXCDATA] section 2.11.1)

The **MetaTagIdsetGiven** property contains a serialization of **REPLGUID**-based **IDSET** structures, as specified in section 2.2.2.4.2. The **IDSETs** contain **Folder ID** structures ([MS-OXCDATA] section 2.2.1.1) for hierarchy synchronization operations, or **Message ID** structures for content synchronization operations, that exist in the local replica of the client. The **IDSETs** MUST NOT include any IDs that are not in the local replica of the client. Because of this restriction on IDs, this property often does not compress as well as the **MetaTagCnset*** properties, which makes the **MetaTagIdsetGiven** property grow larger than the **MetaTagCnset*** properties. For more details about compression of **IDSETs**, see section 3.1.5.5.

For more details about sending and receiving this property, see sections 3.2.5.2.1 and 3.3.5.13.

2.2.1.1.2 MetaTagCnsetSeen ICS State Property

Property ID: 0x6796

Data type: **PtypBinary**, 0x0102 ([MS-OXCDATA] section 2.11.1)

The **MetaTagCnsetSeen** property contains a serialization of **REPLGUID**-based **CNSET** structures, as specified in section 2.2.2.4. The **CN** structures, as specified in section 2.2.2.1, in the **CNSET** track changes to folders (for hierarchy synchronization operations) or **normal messages** (for content synchronization operations) in the current synchronization scope that have been previously communicated to a client, and are reflected in its local replica.

2.2.1.1.3 MetaTagCnsetSeenFAI ICS State Property

Property ID: 0x67DA

Data type: **PtypBinary**, 0x0102 ([MS-OXCDATA] section 2.11.1)

The **MetaTagCnsetSeenFAI** property contains a serialization of **REPLGUID**-based **IDSET** structures, as specified in section 2.2.2.4. The semantics of this property are identical to the **MetaTagCnsetSeen** property (section 2.2.1.1.2), except that this property contains IDs for **folder associated information (FAI)** messages and is therefore only used in content synchronization operations.

2.2.1.1.4 MetaTagCnsetRead ICS State Property

Property ID: 0x67D2

Data type: **PtypBinary**, 0x0102 ([MS-OXCDATA] section 2.11.1)

The **MetaTagCnsetRead** property contains a serialization of **REPLGUID**-based **CNSET** structures, as specified in section 2.2.2.4. The **CN** structures, as specified in section 2.2.2.1, in the **CNSET** track changes to the read state for messages in the current synchronization scope that have been previously communicated to the client and are reflected in its local replica. This property does not track whether messages have been read, it only tracks changes to the read state of a message. For more details about tracking read state changes, see section 3.2.5.6.

The read state of a message is determined by the **PidTagMessageFlags** property ([MS-OXCMSG] section 2.2.1.6), which contains a bitmask of flags that indicate the origin and current state of the message.

2.2.1.2 Messaging Object Identification and Change Tracking Properties

This section specifies details about the properties that are used by this protocol to identify messages, folders, and track changes.

For details about how messaging object and change identification values are created and modified by the protocol roles, see section 3.1.5.3.

2.2.1.2.1 PidTagMid Property

Data type: **PtypInteger64** ([MS-OXCDATA] section 2.11.1)

The **PidTagMid** property ([MS-OXPROPS] section 2.792) contains the **MID** structure ([MS-OXCDATA] section 2.2.1.2) of the message currently being synchronized.

For details about the presence of the **PidTagMid** property in message change headers, see the **SynchronizationExtraFlags** field in section 2.2.3.2.1.1.1.

2.2.1.2.2 PidTagFolderId Property

Data type: **PtypInteger64** ([MS-OXCDATA] section 2.11.1)

The **PidTagFolderId** property ([MS-OXPROPS] section 2.691) contains the **Folder ID** structure ([MS-OXCDATA] section 2.2.1.1) of the folder currently being synchronized.

For details about the presence of the **PidTagFolderId** property in message change headers, see the **SynchronizationExtraFlags** field in section 2.2.3.2.1.1.1.

2.2.1.2.3 PidTagChangeNumber Property

Data type: **PtypInteger64** ([MS-OXCDATA] section 2.11.1)

The **PidTagChangeNumber** property ([MS-OXPROPS] section 2.623) contains the **CN** structure, as specified in section 2.2.2.1, that identifies the last change to the message or folder that is currently being synchronized.

For details about the presence of the **PidTagChangeNumber** property in message change headers, see the **SynchronizationExtraFlags** field in section 2.2.3.2.1.1.1.

2.2.1.2.4 PidTagParentFolderId Property

Data type: **PtypInteger64** ([MS-OXCDATA] section 2.11.1)

The **PidTagParentFolderId** property ([MS-OXPROPS] section 2.850) contains the **Folder ID** structure ([MS-OXCDATA] section 2.2.1.1) that identifies the parent folder of the messaging object being synchronized.

2.2.1.2.5 PidTagSourceKey Property

Data type: **PtypBinary** ([MS-OXCDATA] section 2.11.1)

The **PidTagSourceKey** property ([MS-OXPROPS] section 2.1012) contains a serialized **XID** structure, as specified in section 2.2.2.2, that specifies the **internal identifier** for the folder or message.

For more details about how clients generate this property, see section 3.3.5.2.1. For more details about how servers generate or output this property value, see section 3.2.5.5.

2.2.1.2.6 PidTagParentSourceKey Property

Data type: **PtypBinary** ([MS-OXCDATA] section 2.11.1)

The **PidTagParentSourceKey** property ([MS-OXPROPS] section 2.852) specifies the **PidTagSourceKey** property (section 2.2.1.2.5) of the current folder's parent folder.

2.2.1.2.7 PidTagChangeKey Property

Data type: **PtypBinary** ([MS-OXCDATA] section 2.11.1)

The **PidTagChangeKey** property ([MS-OXPROPS] section 2.622) contains a serialized **XID** structure, as specified in section 2.2.2.2, that identifies the last change to the messaging object.

If the last change to the messaging object was imported from the client by using the **RopSynchronizationImportMessageChange** ROP (section 2.2.3.2.4.2), the value of the **PidTagChangeKey** property that is saved to the **message store** by the server contains the value for the **PidTagChangeKey** property that was passed in the **PropertyValues** field of the **RopSynchronizationImportMessageChange** ROP request buffer.

If the last change to a messaging object was made by the server, the value of the **PidTagChangeKey** property that is saved to the message store by the server contains an **XID** generated from the **PidTagChangeNumber** property (section 2.2.1.2.3). For more details about generating **XIDs** based on internal identifiers, see section 3.1.5.3.

2.2.1.2.8 PidTagPredecessorChangeList Property

Data type: **PtypBinary** ([MS-OXCDATA] section 2.11.1)

The **PidTagPredecessorChangeList** property ([MS-OXPROPS] section 2.858) contains **PredecessorChangeList** structures, as specified in section 2.2.2.3. The **PidTagPredecessorChangeList** property contains all the **XID** structures, as specified in section 2.2.2.2, from all replicas (1) that have been integrated into the current version of the messaging object. This property is used in **conflict detection** by all protocol roles.

2.2.1.2.9 PidTagOriginalEntryId Property

Data type: **PtypBinary** ([MS-OXCDATA] section 2.11.1)

The **PidTagOriginalEntryId** property ([MS-OXPROPS] section 2.820) contains the original EntryID of the message, which is used to associate the full message item being downloaded from the server with the message header previously stored on the client.

2.2.1.3 Meta-Properties for Encoding Differences in Replica Content

Because servers do not maintain a per-client state, the following properties are not persisted on servers and are only present as data in the FastTransfer streams.

All properties are of the **PtypBinary** ([MS-OXCDATA] section 2.11.1) type, and contain a serialized **IDSET** in the **replica ID (REPLID)**-based form, as specified in section 2.2.2.4.1.

2.2.1.3.1 MetaTagIdsetDeleted Meta-Property

Property ID: 0x67E5

Data type: **PtypBinary**, 0x0102 ([MS-OXCDATA] section 2.11.1)

The **MetaTagIdsetDeleted** property contains a serialization of a **REPLID**-based **IDSET** structures, as specified in section 2.2.2.4.1. The **IDSETs** contain the IDs of folders (for hierarchy synchronization operations) or messages (for content synchronization operations) that were **hard deleted** or **soft deleted** since the last synchronization identified by the initial ICS state. For more details about how an **IDSET** is serialized, see section 3.1.5.4.

2.2.1.3.2 MetaTagIdsetNoLongerInScope Meta-Property

Property ID: 0x4021

Data type: **PtypBinary**, 0x0102 ([MS-OXCDATA] section 2.11.1)

The **MetaTagIdsetNoLongerInScope** property contains a serialization of a **REPLID**-based **IDSET** structures, as specified in section 2.2.2.4.1. The **IDSETs** contain the IDs of messages that got out of the synchronization scope since the last synchronization identified by the initial ICS state. Messages that no longer match a restriction are considered out of synchronization scope. For more details about how an **IDSET** is serialized, see section 3.1.5.4.

Note that messages moved to another folder are considered soft deleted in the source folder; hard deleted and soft deleted messages are reported in the **MetaTagIdsetDeleted** property (section 2.2.1.3.1).

2.2.1.3.3 MetaTagIdsetExpired Meta-Property

Property ID: 0x6793

Data type: **PtypBinary**, 0x0102 ([MS-OXCDATA] section 2.11.1)

The **MetaTagIdsetExpired** property<1> contains a serialization of **REPLID**-based **IDSET** structures. The **IDSETs** contain IDs of **expired Message objects** in a **public folder** that expired since the last synchronization identified by the initial ICS state. For more details about how an **IDSET** is serialized, see section 3.1.5.4.

2.2.1.3.4 MetaTagIdsetRead Meta-Property

Property ID: 0x402D

Data type: **PtypBinary**, 0x0102 ([MS-OXCDATA] section 2.11.1)

The **MetaTagIdsetRead** property contains a serialization of **REPLID**-based **IDSET** structures. The **IDSETs** contain IDs of messages that were marked as read (as specified by the **PidTagMessageFlags** property in [MS-OXCMSG] section 2.2.1.6) since the last synchronization, as identified by the initial ICS state. For more details about how an **IDSET** is serialized, see section 3.1.5.4.

2.2.1.3.5 MetaTagIdsetUnread Meta-Property

Property ID: 0x402E

Data type: **PtypBinary**, 0x0102 ([MS-OXCDATA] section 2.11.1)

The **MetaTagIdsetUnread** property contains a serialization of **REPLID**-based **IDSET** structures. The **IDSETs** contain IDs of messages that were marked as unread (as specified by the **PidTagMessageFlags** property in [MS-OXCMSG] section 2.2.1.6) since the last synchronization, as identified by the initial ICS state. For more details about how an **IDSET** is serialized, see section 3.1.5.4.

2.2.1.4 Conflict Resolution Properties

This section specifies details about the properties that are used in conflict resolution.

2.2.1.4.1 PidTagResolveMethod Property

Data type: **PtypInteger32** ([MS-OXCDATA] section 2.11.1)

The **PidTagResolveMethod** property ([MS-OXPROPS] section 2.920) specifies how to resolve any conflicts with the message. This property is not required.

The following table defines valid values for the **PidTagResolveMethod** property.

Flag name	Value	Description
RESOLVE_METHOD_DEFAULT	0x00000000	A conflict resolve message SHOULD be generated.
RESOLVE_METHOD_LAST_WRITER_WINS	0x00000001	Overwrite the target message with the current changes being applied.
RESOLVE_NO_CONFLICT_NOTIFICATION	0x00000002	Do not send a conflict notification message when generating a conflict resolve message in a public folder.

For more details about **conflict resolution**, see section 3.1.5.6.

2.2.1.4.2 PidTagConflictEntryId Property

Data type: **PtypBinary** ([MS-OXCDATA] section 2.11.1)

The **PidTagConflictEntryId** property ([MS-OXPROPS] section 2.632) contains the **EntryID** of the conflict resolve message, as specified in section 3.1.5.6.2.1.

2.2.1.4.3 PidTagInConflict Property

Data type: **PtypBoolean** ([MS-OXCDATA] section 2.11.1)

The **PidTagInConflict** property ([MS-OXPROPS] section 2.730) specifies whether the attachment represents an alternate replica.

2.2.1.5 PidTagAssociated Property

Data type: **PtypBoolean** ([MS-OXCDATA] section 2.11.1)

The **PidTagAssociated** property ([MS-OXPROPS] section 2.575) specifies whether the message being synchronized is an FAI message.

2.2.1.6 PidTagMessageSize Property

Data type: **PtypInteger32** ([MS-OXCDATA] section 2.11.1)

The **PidTagMessageSize** property ([MS-OXPROPS] section 2.787) identifies the size of the message in bytes.

For details about the presence of the **PidTagMessageSize** property in message change headers, see section 2.2.3.2.1.1.1.

2.2.1.7 Properties That Denote Subobjects

The properties in the following tables denote subobjects of the messaging objects and can be used in the following:

- The property inclusion and exclusion lists of ROPs that configure download operations. For example, the **RopSynchronizationConfigure** ROP (section 2.2.3.2.1.1) and the **RopFastTransferSourceCopyTo** ROP (section 2.2.3.1.1.1) both configure download operations.
- As values of **MetaTagFXDelProp** meta-properties, as specified in section 2.2.4.1.5.1.

Folder properties	Description
PidTagContainerContents ([MS-OXPROPS] section 2.634)	Identifies all normal messages in the current folder.
PidTagFolderAssociatedContents ([MS-OXPROPS] section 2.690)	Identifies all FAI messages in the current folder.
PidTagContainerHierarchy ([MS-OXPROPS] section 2.636)	Identifies all subfolders of the current folder. Clients use this property in inclusion and exclusion lists, but do not use this property as a value of the **MetaTagFXDelProp** **meta-property**.

Message properties	Description
PidTagMessageRecipients ([MS-OXPROPS] section 2.786)	Identifies all recipients of the current message.
PidTagMessageAttachments ([MS-OXPROPS] section	Identifies all attachments to the current

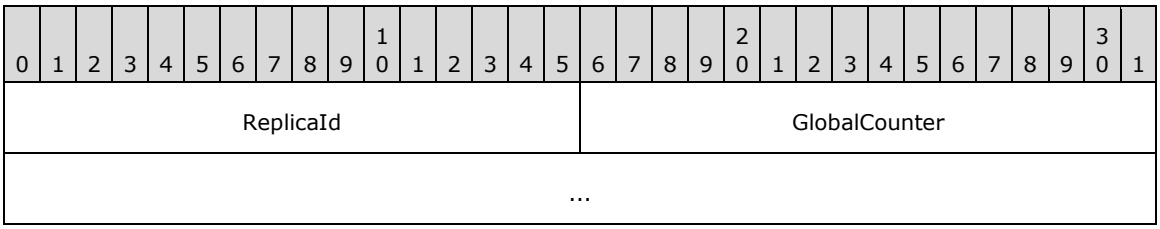
Message properties	Description
2.776)	message.

Attachment properties	Description
PidTagAttachDataObject ([MS-OXCMSG] section 2.2.2.8)	Identifies the **Embedded Message object** of the current attachment. Clients do not use this property as a value of the **MetaTagFXDelProp** meta-property.

2.2.2 Structures

2.2.2.1 CN Structure

A **CN** structure contains a **change number** that identifies a version of a messaging object. **CNs** are identical in format to **Folder ID** structures ([MS-OXCDATA] section 2.2.1.1) and **Message ID** structures ([MS-OXCDATA] section 2.2.1.2), except the **GlobalCounter** field represents a change to a messaging object rather than a messaging object itself.

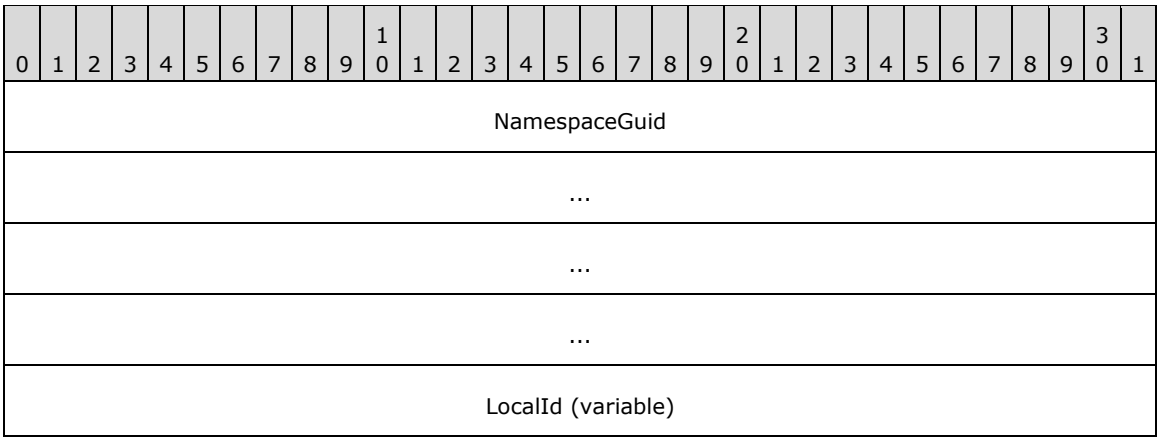


ReplicaId (2 bytes): A 16-bit unsigned integer identifying the **server replica** in which the messaging object was last changed.

GlobalCounter (6 bytes): An unsigned 48-bit integer identifying the change to the messaging object.

2.2.2.2 XID Structure

An **XID** structure contains an **external identifier** for an entity within a message store.



...

NamespaceGuid (16 bytes): A 128-bit **GUID**. This field identifies the namespace of the **LocalId** field.

LocalId (variable): A variable binary value. This field contains the ID of the entity in the namespace specified by the **NamespaceGuid** field. This field has a minimum length of 1 byte and a maximum length of 8 bytes.

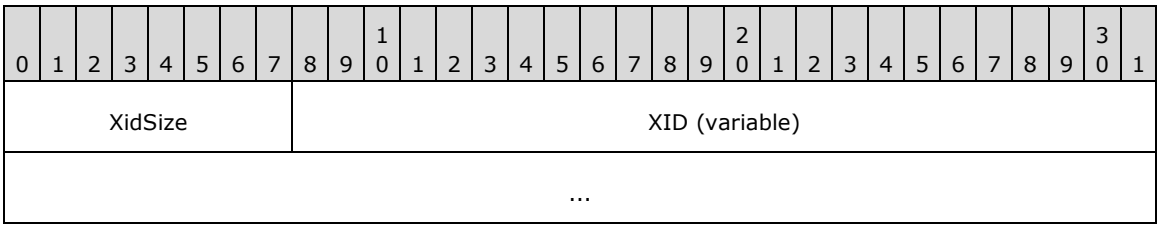
For more details about **GID** structures, which are a subtype of an **XID** structure, see [MS-OXCDATA] section 2.2.1.3. For **GIDs**, the **DatabaseGuid** field maps to the **NamespaceGuid** field, and the **GlobalCounter** field maps to the **LocalId** field.

All **XID** structures that have the same value for their **NamespaceGuid** fields MUST have **LocalId** fields of the same length. However, the size of the value specified by the **LocalId** field cannot be determined by examining the value of the **NamespaceGuid** field and MUST be provided externally. In most cases, **XIDs** are present within other structures that specify information about the size of the **XID**, such as the **SizedXid** structure, as specified in section 2.2.2.3.1, or the **propValue** element, as specified in section 2.2.4.3.21.

2.2.2.3 PredecessorChangeList Structure

The **PredecessorChangeList** structure contains a set of **XID** structures, as specified in section 2.2.2.2, that identify change numbers of messaging objects in different replicas (1). The order of the **XIDs** does not have significance for interpretation, but is significant for serialization and deserialization. The set of **XIDs** MUST be serialized without padding as an array of **SizedXid** structures binary-sorted by the value of **NamespaceGuid** field of the **XID** structure in the ascending order.

2.2.2.3.1 SizedXid Structure



XidSize (1 byte): An unsigned 8-bit integer that specifies the length of the **XID** field, in bytes.

XID (variable): An **XID** structure, as specified in section 2.2.2.2, that contains the value of the internal identifier of an object, or internal or external identifier of a change number. The length of this field is specified by the **XidSize** field, in bytes.

2.2.2.4 IDSET and CNSET Structures

An **IDSET** structure contains a set of ID values. The ID values are one of the following types:

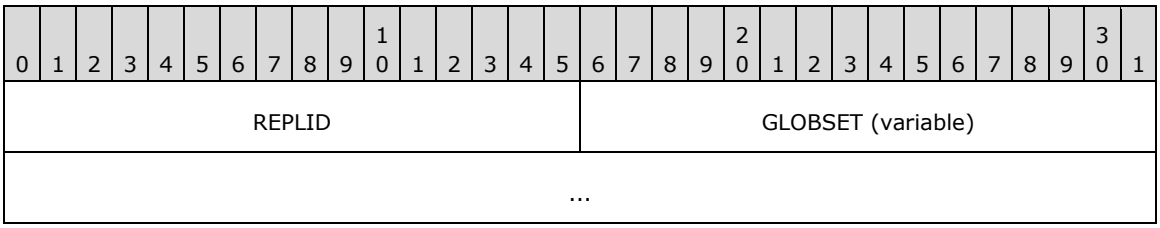
- **Message ID** structures ([MS-OXCDATA] section 2.2.1.2).
- **Folder ID** structures ([MS-OXCDATA] section 2.2.1.1).
- **CN** structures, as specified in section 2.2.2.1.

When an **IDSET** structure contains **CNs**, it is also known as a **CNSET** structure. In this section, the term **IDSET** is used to refer to both **IDSET** and **CNSET** structures.

The **IDSET** serialization format specified in the following sections is optimized for data transfer, and is not intended for in-memory operations. For details about the serialization and deserialization process, see section 3.1.5.4.

2.2.2.4.1 Serialized IDSET Structure Containing a REPLID Structure

For every **REPLID** and **GLOBSET** structure pair represented in the formatted **IDSET** structure, add the following values to the serialization buffer in lowest to highest **REPLID** structure order. **GLOBSET** structures are defined in section 2.2.2.6.

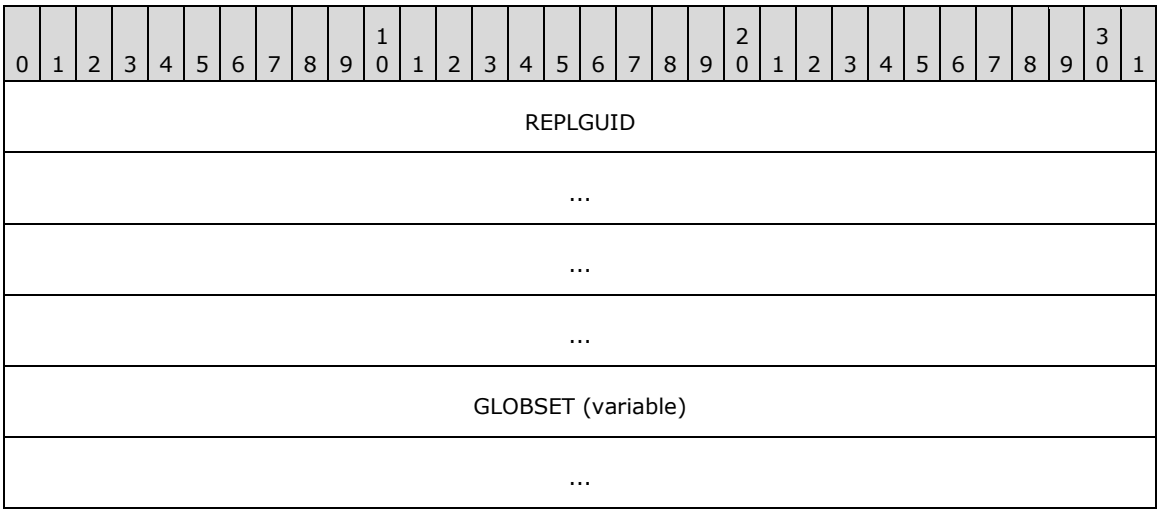


REPLID (2 bytes): A **REPLID** structure that when combined with all **GLOBCNT** structures contained in the **GLOBSET** field, produces a set of IDs.

GLOBSET (variable): A serialized **GLOBSET** structure.

2.2.2.4.2 Serialized IDSET Structure Containing a REPLGUID Structure

For every **REPLGUID** and **GLOBSET** structure pair represented in the formatted **IDSET** structure, add the following to the serialization buffer. **REPLGUID**-**GLOBSET** structure pairs MUST be serialized by the value of the **REPLGUID** in the ascending order, using byte-to-byte comparison. **GLOBSET** structures are defined in section 2.2.2.6.



REPLGUID (16 bytes): A GUID that identifies a **REPLGUID** structure. When this GUID is combined with the values of the **GLOBCNT** structures contained in the **GLOBSET** field, it produces a set of **GID** structures ([MS-OXCDATA] section 2.2.1.3). The GUID value can also be converted into a **REPLID** structure to produce a set of **Message ID** ([MS-OXCDATA] section 2.2.1.2) or **Folder ID** ([MS-OXCDATA] section 2.2.1.1) structures.

GLOBSET (variable): A serialized **GLOBSET** structure.

2.2.2.5 GLOBCNT Structure

A **GLOBCNT** structure is a 6-byte global namespace counter. If a **GLOBCNT** is paired with a **REPLID** structure it forms a **Message ID** ([MS-OXCDATA] section 2.2.1.2), **Folder ID** ([MS-OXCDATA] section 2.2.1.1), or **CN** structure as specified in section 2.2.2.1. If a **GLOBCNT** is paired with a **REPLGUID** structure it forms a **GID** structure ([MS-OXCDATA] section 2.2.1.3).

2.2.2.6 GLOBSET Structure

A **GLOBSET** structure is a set of **GLOBCNT** structures, as specified in section 2.2.2.5, that are reduced to one or more **GLOBCNT** ranges. A single **GLOBCNT** range identifies only the lowest and highest values in a set of consecutive **GLOBCNT** values. A **GLOBCNT** range is created using any of the commands in this section, with the exception of the **Pop** and **End** commands.

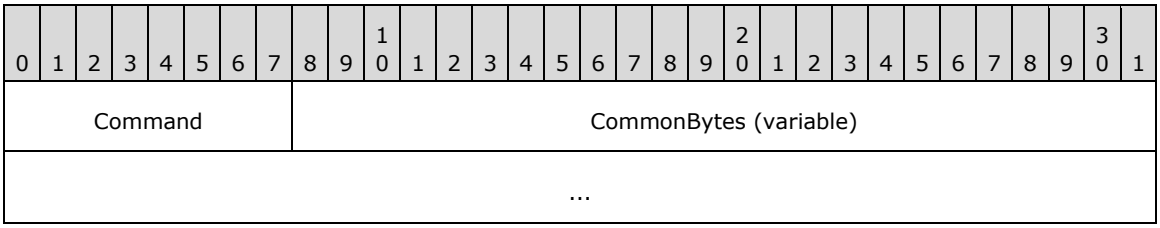
The serialization format specified in the following sections is optimized for data transfer, and is not intended for in-memory operations.

A **GLOBSET** is serialized without padding as a set of commands. For details about how to encode or decode a **GLOBSET** by using the commands in this section, see section 3.1.5.4.3.

2.2.2.6.1 Push Command (0x01 – 0x06)

The **Push** command places high-order bytes onto the **common byte stack**.

For information on how a **GLOBSET** structure, as specified in section 2.2.2.6, is encoded using this command, see section 3.1.5.4.3.1.1. For information on how a **GLOBSET** structure is decoded using this command, see section 3.1.5.4.3.2.1.



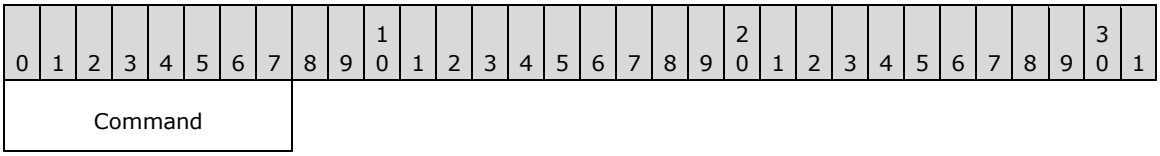
Command (1 byte): An integer that specifies the number of high-order bytes that the **GLOBCNT** structures, as specified in section 2.2.2.5, share. This value MUST be in the range 0x01 through 0x06.

CommonBytes (variable): A byte array that contains the bytes shared by the **GLOBCNT** structures, as specified in section 2.2.2.5, that are pushed onto the common byte stack. The length of this field is specified by value of the **Command** field (0x01 through 0x06), in bytes.

2.2.2.6.2 Pop Command (0x50)

The **Pop** command removes bytes that were added to the common byte stack from the previous **Push** command, as specified in section 2.2.2.6.1.

For information about how a **GLOBSET** structure, as specified in section 2.2.2.6, is encoded using this command, see section 3.1.5.4.3.1.2. For information about how a **GLOBSET** structure is decoded using this command, see section 3.1.5.4.3.2.2.

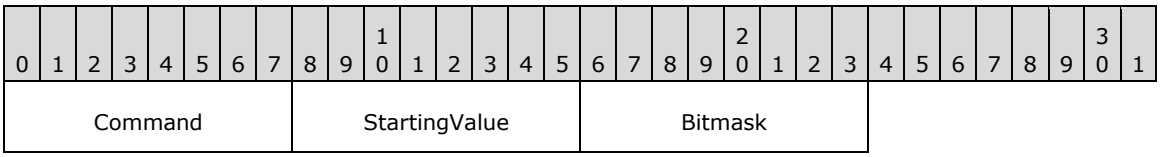


Command (1 byte): This value MUST be set to0x50.

2.2.2.6.3 Bitmask Command (0x42)

The **Bitmask** command compresses up to five **GLOBCNT** ranges, as specified in section 2.2.2.6, into a single encoding command if they all have 5 high-order bytes in common and the low-order bytes are all within eight values of each other.

For information about how a **GLOBSET** structure, as specified in section 2.2.2.6, is encoded using this command, see section 3.1.5.4.3.1.3. For information about how a **GLOBSET** structure is decoded using this command, see section 3.1.5.4.3.2.3.



Command (1 byte): This value MUST be set to 0x42.

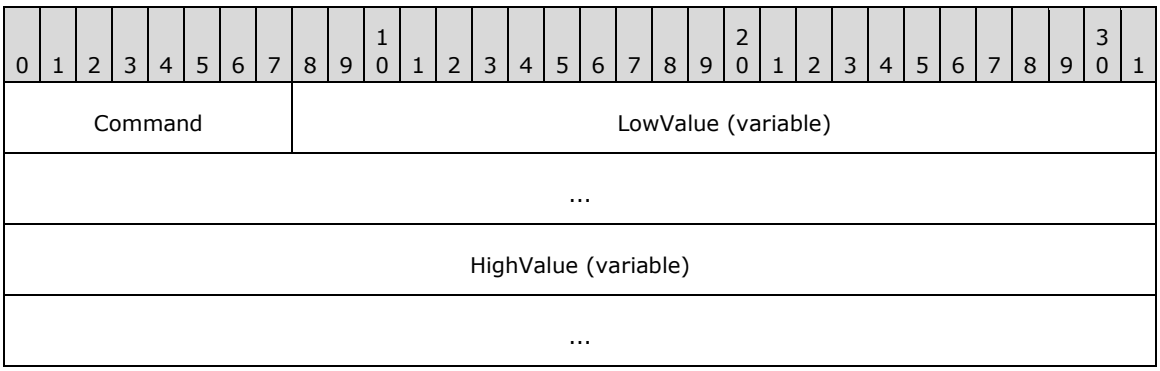
StartingValue (1 byte): The low-order byte of the first **GLOBCNT** structure, as specified in section 2.2.2.5.

Bitmask (1 byte): A flag that identifies whether the **GLOBCNT** structure that exists in the **GLOBCNT** range that starts with the value of the **StartingValue** field.

2.2.2.6.4 Range Command (0x52)

The **Range** command adds a **GLOBCNT** range to the **GLOBSET** structure, as specified in section 2.2.2.6. The range is determined by the **GLOBCNT** structure produced from the **LowValue** field and the **GLOBCNT** structure produced from the **HighValue** field.

For more details about how a **GLOBSET** structure, as specified in section 2.2.2.6, is encoded using this command, see section 3.1.5.4.3.1.4. For details about how a **GLOBSET** structure is decoded using this command, see section 3.1.5.4.3.2.4.



Command (1 byte): This value MUST be set to 0x52.

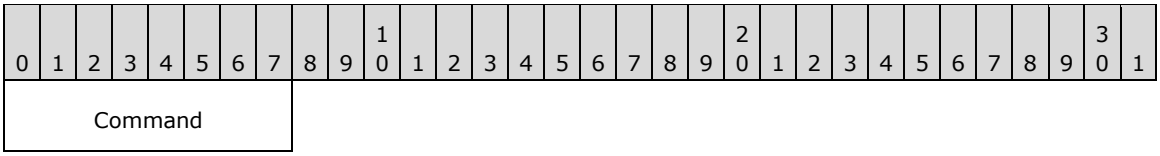
LowValue (variable): A byte array of low-order values for **GLOBCNT** structure generation. The number of bytes in this field is equal to 6 minus the number of high-order bytes in the common byte stack. This value MUST be less than or equal to the value of the **HighValue** field.

HighValue (variable): A byte array of low-order values for **GLOBCNT** structure generation. The number of bytes in this field is equal to 6 minus the number of high-order bytes in the common byte stack. This value MUST be greater than or equal to the value of the **LowValue** field.

2.2.2.6.5 End Command (0x00)

The **End** command is used to signal the end of the **GLOBSET** structure encoding.

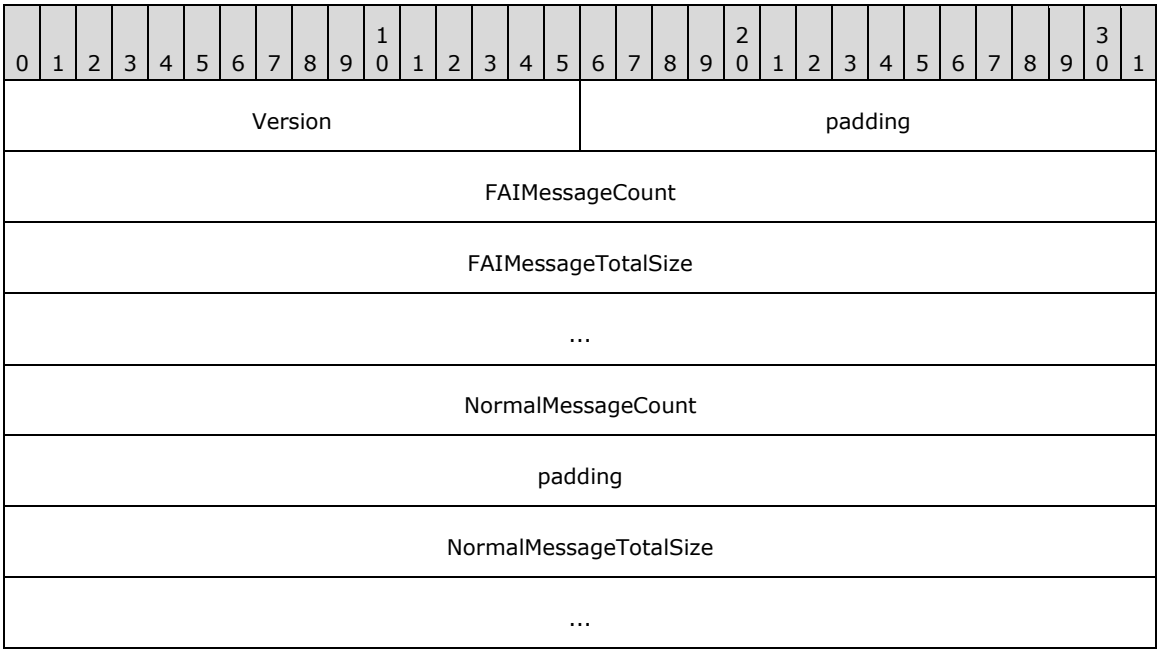
For information about how a **GLOBSET** structure, as specified in section 2.2.2.6, is encoded using this command, see section 3.1.5.4.3.1.5. For information about how a **GLOBSET** structure is decoded using this command, see section 3.1.5.4.3.2.5.



Command (1 byte): This value MUST be set to 0x00.

2.2.2.7 ProgressInformation Structure

The **ProgressInformation** structure is used by the **progressTotal** element, as specified in section 2.2.4.3.19, to describe the approximate size of all the **messageChange** elements, as specified in section 2.2.4.3.11, that follow in the FastTransfer stream.



Version (2 bytes): An unsigned 16-bit value that contains a number that identifies the binary structure of the data that follows. The preceding packet diagram specifies the format for version 0x0000, which is the only version of this structure defined for this protocol.

padding (2 bytes): This value SHOULD be set to 0x0000 and MUST be ignored by clients.

FAIMessageCount (4 bytes): An unsigned 32-bit integer value that contains the total number of changes to FAI messages that are scheduled for download during the current synchronization operation.

FAIMessageTotalSize (8 bytes): An unsigned 64-bit integer value that contains the size in bytes of all changes to FAI messages that are scheduled for download during the current synchronization operation.

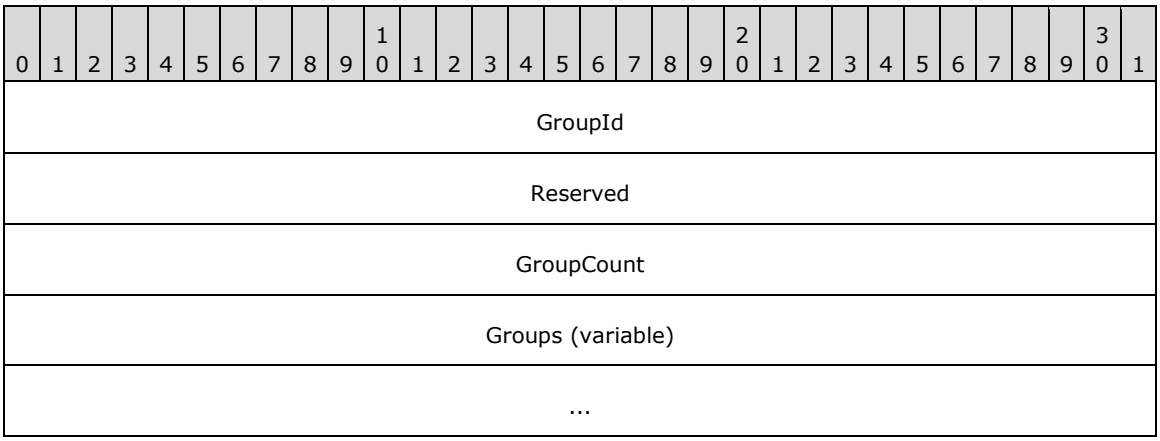
NormalMessageCount (4 bytes): An unsigned 32-bit integer value that contains the total number of changes to normal messages that are scheduled for download during the current synchronization operation.

padding (4 bytes): This value SHOULD be set to 0x00000000 and MUST be ignored by clients.

NormalMessageTotalSize (8 bytes): An unsigned 64-bit integer value that contains the size in bytes of all changes to normal messages that are scheduled for download during the current synchronization operation.

2.2.2.8 PropertyGroupInfo Structure

The **PropertyGroupInfo** structure specifies a mapping between a group index and **property tags** within a property group. For more details about property groups, see section 3.2.5.7.



GroupId (4 bytes): An unsigned 32-bit integer value that identifies a property mapping within the current synchronization download context.

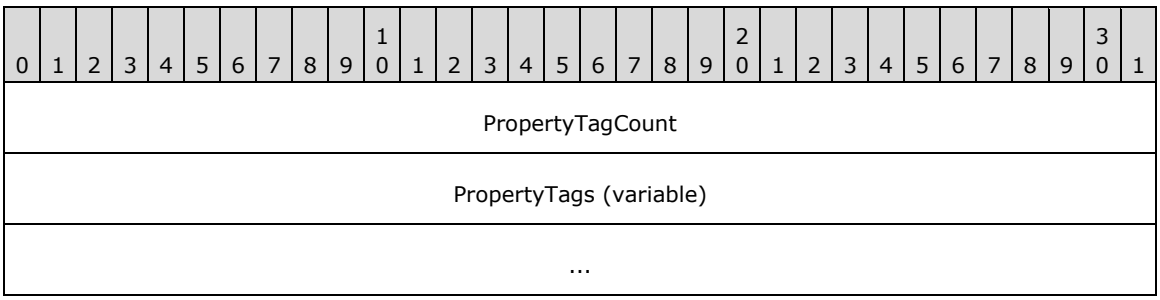
Reserved (4 bytes): This value MUST be set to 0x00000000.

GroupCount (4 bytes): An unsigned 32-bit integer value that specifies how many **PropertyGroup** structures, as specified in section 2.2.2.8.1, are present in the **Groups** field. This field MUST NOT be set to 0x00000000.

Groups (variable): An array of **PropertyGroup** structures, as specified in section 2.2.2.8.1. The number of **PropertyGroup** structures in this value is specified by the value of the **GroupCount** field.

2.2.2.8.1 PropertyGroup Structure

The **PropertyGroup** structure specifies the property tags that belong to the property group for use in the **PropertyGroupInfo** structure, as specified in section 2.2.2.8.

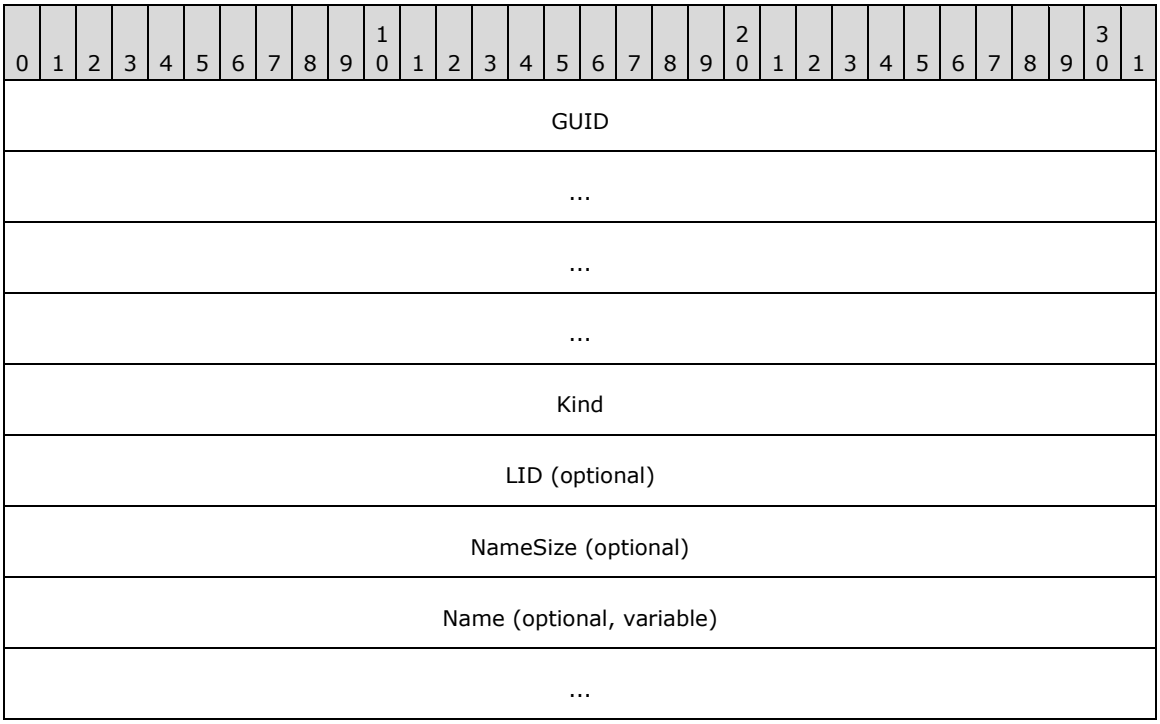


PropertyTagCount (4 bytes): An unsigned 32-bit integer value that specifies how many **PropertyTag** structures are present in the **PropertyTags** field. This value MUST NOT be set to 0x00000000.

PropertyTags (variable): An array of **PropertyTag** structures ([MS-OXCDATA] section 2.9). If a **PropertyTag** structure identifies a **named property**, the **PropertyTag** is immediately followed by a **GroupPropertyName** structure, as specified in section 2.2.2.8.1.1. Named properties are identified by a **PropertyId** structure ([MS-OXCDATA] section 2.9) with a value greater than or equal to 0x8000. The number of **PropertyTag** structures in this field is specified by the value of the **PropertyTagCount** field.

2.2.2.8.1.1 GroupPropertyName Structure

The **GroupPropertyName** structure defines the named property included in the **PropertyGroup** structure, as specified in section 2.2.2.8.1.



GUID (16 bytes): The GUID that identifies the **property set** for the named properties.

Kind (4 bytes): A value that identifies the type of property. The following table lists the possible values for the **Kind** field:

Name	Value
0x00000000	The property is identified by the LID field.
0x00000001	The property is identified by the Name field.

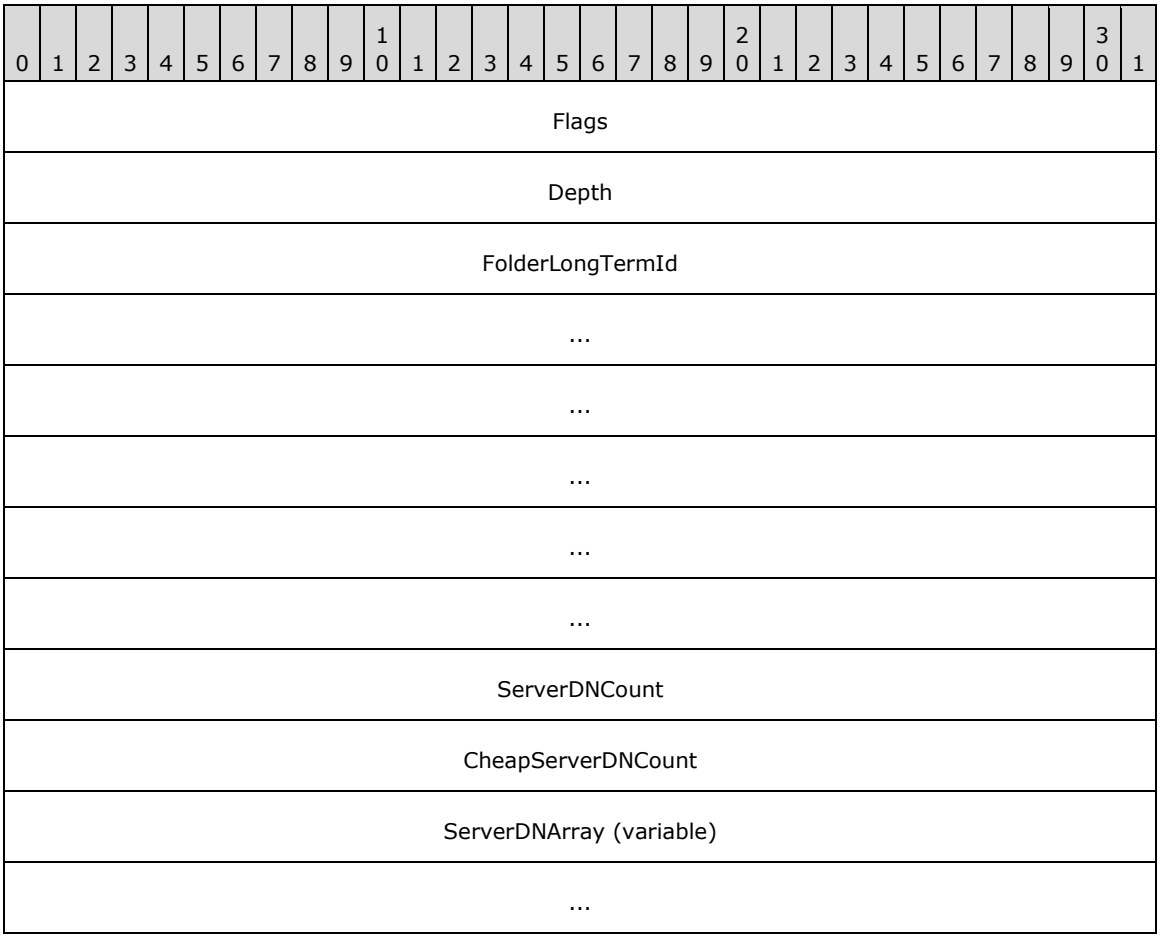
LID (optional) (4 bytes): A value that identifies the named property within its property set. This value is present only if the **Kind** field is set to 0x00000000.

NameSize (optional) (4 bytes): A value that specifies the length of the **Name** field, in bytes. This value is present only if the **Kind** field is set to 0x00000001.

Name (optional, variable): A **Unicode** (UTF-16) string that identifies the property within the property set. This value is present only if the **Kind** field is set to 0x00000001. The length of this field is specified by the value of the **NameSize** field, in bytes.

2.2.2.9 FolderReplicaInfo Structure

The **FolderReplicaInfo** structure contains information about server replicas of a public folder.



Flags (4 bytes): This value MUST be set to 0x00000000.

Depth (4 bytes): This value MUST be set to 0x00000000.

FolderLongTermId (24 bytes): A LongTermID structure ([MS-OXCDATA] section 2.2.1.3.1) that identifies the folder for which the server replica information is being described.

ServerDNCount (4 bytes): An unsigned integer value that determines how many elements exist in the **ServerDNArray** field. This value MUST NOT be 0x00000000.

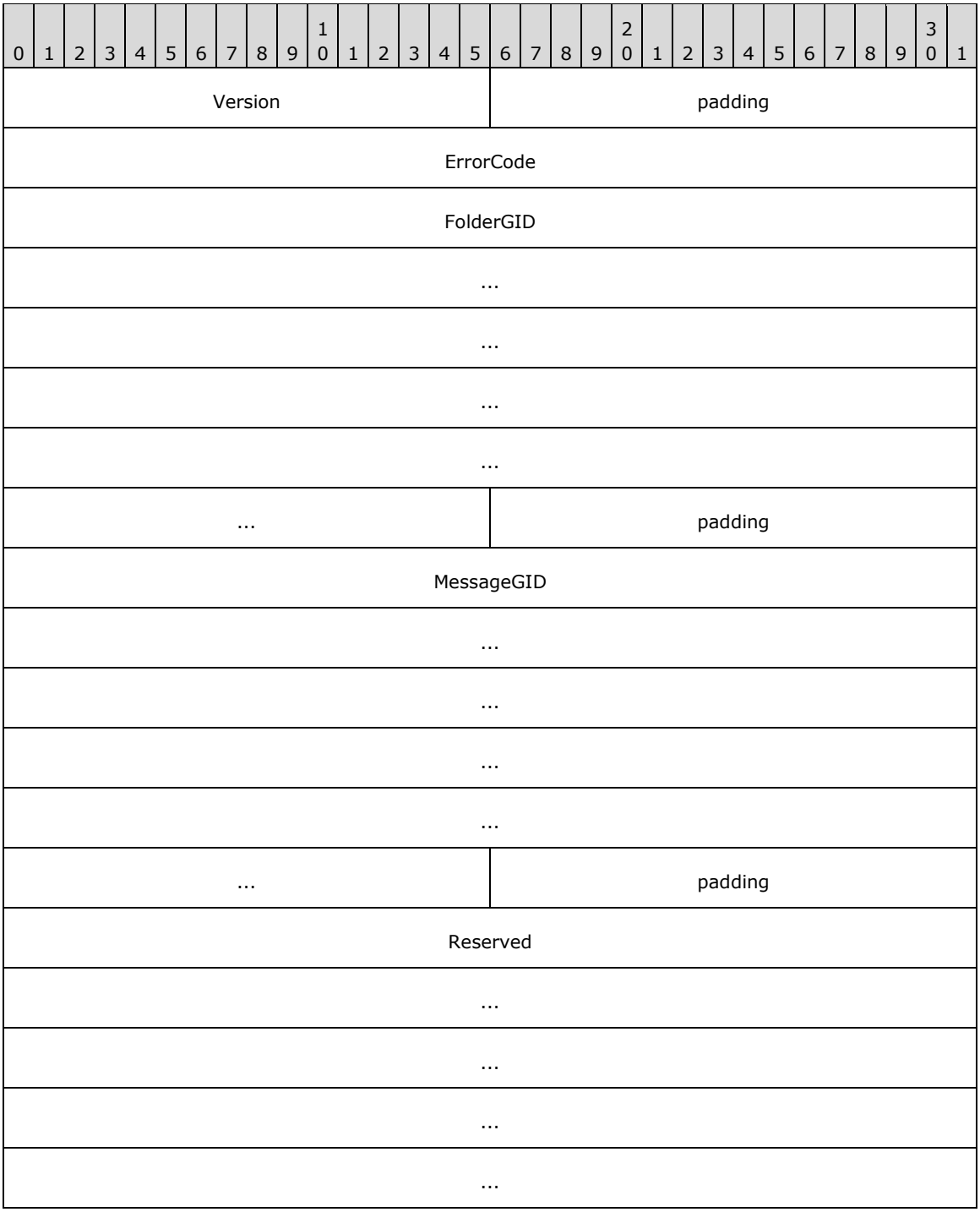
CheapServerDNCount (4 bytes): An unsigned integer value that determines how many of the leading elements in the **ServerDNArray** field have the same, lowest, network access cost. The value of the **CheapServerDNCount** field MUST be less than or equal to value of the **ServerDNCount** field.

ServerDNArray (variable): An array of **ASCII**-encoded NULL-terminated strings. This value contains an **enterprise/site/server distinguished name (ESSDN)** of servers that have a replica (1) of the folder identified by the value of the **FolderLongTermId** field. The number of **ServerDNCount** strings in this field is specified by the **ServerDNCount** field.

2.2.2.10 ExtendedErrorInfo Structure

The **ExtendedErrorInfo** structure contains extended and contextual information about an error that occurred when producing a FastTransfer stream.

For details about how this structure is used in FastTransfer error recovery and reporting of **partial completion** of download operations, see section 2.2.4.3.4.



...
AuxBytesCount
AuxBytesOffset
Reserved (variable)
...
AuxBytes (variable)
...

Version (2 bytes): An unsigned 16-bit integer that determines the format of the structure. The format shown in the preceding packet diagram corresponds to version 0x00000000, which is the only version defined for the protocol.

padding (2 bytes): This value SHOULD be set to 0x0000 and MUST be ignored by clients.

ErrorCode (4 bytes): An error codes from the Data Structures Protocol, as specified in [MS-OXCDATA] section 2.4, that describes the reason for the failure.

FolderGID (22 bytes): A **GID** structure ([MS-OXCDATA] section 2.2.1.3) that identifies the folder that was in context at the time the error occurred. This value MUST be set to zero, if no folders were in context.

padding (2 bytes): This value SHOULD be set to 0x0000 and MUST be ignored by the clients.

MessageGID (22 bytes): A **GID** structure that identifies the message that was in context at the time the error occurred. This value MUST be set to zero, if no messages were in context.

padding (2 bytes): This value SHOULD be set to 0x0000 and MUST be ignored by clients.

Reserved (24 bytes): This value SHOULD be set to 0x0000 and SHOULD be ignored by clients.

AuxBytesCount (4 bytes): An unsigned 32-bit integer value that specifies the size of the **AuxBytes** field. If this value is set to 0x00000000, the **AuxBytes** field is missing.

AuxBytesOffset (4 bytes): An unsigned 32-bit integer value that specifies the offset in bytes of the **Auxbytes** field from the beginning of the structure.

Reserved (optional, variable): This value SHOULD be set to zero and SHOULD be ignored by clients.

AuxBytes (optional, variable): A **PtypBinary** ([MS-OXCDATA] section 2.11.1) value that MUST be present and located at the offset specified by the value of the **AuxBytesOffset** field from the beginning of the structure, if and only if value of the **AuxBytesCount** field is greater than zero. If this value is present, it consists of opaque diagnostic information returned from the server and MAY be logged by the client.

2.2.3 ROPs

FastTransfer and ICS operations are performed by sending a specific set of ROP requests to the server.

If a ROP name starts with RopSynchronization, it can only be used in ICS operations.

If a ROP name starts with RopFastTransfer, it can be used in FastTransfer operations, and can also be used ICS operations. For more details, see the following table and the ROP details provided in section 2.2.3.1 and section 2.2.3.2.

All FastTransfer and ICS operations can be separated into similar steps:

1. Initialization: Configure an operation and assign it a context, which is used to identify this operation in all subsequent steps.

2. Data transmission: Transmit the messaging object data based on the context configuration.

3. Checkpointing: An optional step in which data that is required for initialization of the next iteration of this operation is downloaded. For more details about checkpointing, see section 3.3.5.6.

4. Release of resources held on a server: Release the context by using the **RopRelease** ROP ([MS-OXCROPS] section 2.2.15.3).

Note that the context in step 1 is not a messaging object, which means that it is not persisted in a mailbox and its lifetime is limited to the lifetime of the handle that is opened for it.

The following table describes the applicability of ROPs for each step of every FastTransfer or ICS operation. See the ROP details in section 2.2.3.1 and section 2.2.3.2 for usage directions.

Operation	Initialization	Data transmission	Checkpointing
FastTransfer download	**RopFastTransferSourceCopyTo** ROP (section 2.2.3.1.1.1) **RopFastTransferSourceCopyProperties** ROP (section 2.2.3.1.1.2) **RopFastTransferSourceCopyMessages** ROP (section 2.2.3.1.1.3) **RopFastTransferSourceCopyFolder** ROP (section 2.2.3.1.1.4) **RopTellVersion** ROP (section 2.2.3.1.1.6)	**RopFastTransferSourceGetBuffer** ROP (section 2.2.3.1.1.5) Mailbox data is encoded into a FastTransfer stream.	Not applicable.
FastTransfer upload	**RopFastTransferDestinationConfigure** ROP (section 2.2.3.1.2.1) **RopTellVersion** ROP (section 2.2.3.1.1.6)	**RopFastTransferDestinationPutBuffer** ROP (section 2.2.3.1.2.2) Mailbox data is encoded into a FastTransfer stream.	Not applicable.
ICS download	**RopSynchronizationConfigure** ROP (section 2.2.3.2.1.1) **RopSynchronizationUploadStateStreamBegin** ROP (section 2.2.3.2.2.1) **RopSynchronizationUploadStateStreamContinue** ROP (section 2.2.3.2.2.2) **RopSynchronizationUploadStateStreamEnd** ROP (section 2.2.3.2.2.3)	**RopFastTransferSourceGetBuffer** ROP (section 2.2.3.1.1.5) Mailbox data is encoded into a FastTransfer stream.	**RopSynchronizationGetTransferState** ([MS-OXCROPS] section 2.2.13.8) and the **RopFastTransferSourceGetBuffer** ([MS-OXCROPS] section 2.2.12.3) ROPs MAY<2> be used
ICS upload	**RopSynchronizationOpenCollector** ROP (section 2.2.3.2.4.1) **RopSynchronizationUploadStateStreamBegin** ROP (section	**RopSynchronizationImportMessageChange** ROP (section 2.2.3.2.4.2) **RopSynchronizationImportHie**	**RopSynchronizationGetTransferState** ROP (section 2.2.3.2.3.1) **RopFastTransferSource**

Operation	Initialization	Data transmission	Checkpointing
	2.2.3.2.2.1) **RopSynchronizationUploadStateStreamContinue** ROP (section 2.2.3.2.2.2) **RopSynchronizationUploadStateStreamEnd** ROP (section 2.2.3.2.2.3)	**rarchyChange** ROP (section 2.2.3.2.4.3) **RopSynchronizationImportMessageMove** ROP (section 2.2.3.2.4.4) **RopSynchronizationImportDeletes** ROP (section 2.2.3.2.4.5) **RopSynchronizationImportReadStateChanges** ROP (section 2.2.3.2.4.6) ROPs that operate on a Message object.	**GetBuffer** ROP (section 2.2.3.1.1.5)

Whenever the applicability of a ROP or protocol details are discussed in this specification, operations to which an explanation applies are usually referenced by mentioning the type of the context, as specified in the following table.

Context type	Operations it applies to
Download context	FastTransfer download, ICS download
FastTransfer context	FastTransfer download, FastTransfer upload
FastTransfer download context	FastTransfer download
FastTransfer upload context	FastTransfer upload
Synchronization context	ICS download, ICS upload
Synchronization download context	ICS download
Synchronization upload context	ICS upload

For details about the relationship between the **InputHandleIndex** fields defined in this specification and the **InputServerObject** fields defined in the Remote Operation (ROP) List and Encoding Protocol, see [MS-OXCROPS] section 3.2.5.1. For details about the relationship between the **OutputHandleIndex** fields defined in this specification and the **OutputServerObject** fields defined in the ROP List and Encoding Protocol, see [MS-OXCROPS] section 3.2.5.2.

Common values for the **ReturnValue** field are included in ROP responses are specified in [MS-OXCDATA] section 2.4.

2.2.3.1 FastTransfer Copy Operations

2.2.3.1.1 Download

2.2.3.1.1.1 RopFastTransferSourceCopyTo ROP

The **RopFastTransferSourceCopyTo** ROP ([MS-OXCROPS] section 2.2.12.6) initializes a FastTransfer operation to download content from a given messaging object and its descendant subobjects.

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

For details about client behaviors related to this ROP, see sections 3.3.4.1 and section 3.3.4.2.1. For details about server behaviors related to this ROP, see sections 3.2.5.8.1 and 3.2.5.8.1.1.

2.2.3.1.1.1.1 RopFastTransferSourceCopyTo ROP Request Buffer

The following descriptions define valid fields for the request buffer of the **RopFastTransferSourceCopyTo** ROP ([MS-OXCROPS] section 2.2.12.6).

InputServerObject: The value of this field MUST be either an **Attachment object**, Message object, or Folder object.

Level (1 byte): An unsigned 8-bit integer that indicates whether descendant subobjects are copied. Set to 0x00 to copy descendant subobjects. Set to nonzero to exclude all descendant subobjects from being copied. A nonzero value can only be passed if the **InputServerObject** field is a Message object or Folder object. The **Level** field MUST be ignored and treated as if it is set to 0x00 if the **InputServerObject** field is an Attachment object.

CopyFlags (4 byte): A 32-bit flags structure. This structure defines the parameters of the FastTransfer download operation.

The following table defines valid flags for the **CopyFlags** field.

Flag name	Value	Description
Move	0x00000001	This flag MUST only be passed if the value of the **InputServerObject** field is a Folder object or a Message object. If this flag is set, the FastTransfer operation is being configured as a logical part of a larger object move operation, as opposed to a copy operation, and the client will issue further operations such as deleting the moved messages from the source. If this flag is not set, the FastTransfer operation is not being configured as a logical part of a larger object move operation. For more details, see section 3.2.5.8.1.1.
Unused1	0x00000002	This flag MUST be ignored on receipt.
Unused2	0x00000004	This flag MUST be ignored on receipt.
Unused3	0x00000008	This flag MUST be ignored on receipt.
Unused4	0x00000200	This flag MUST be ignored on receipt.
Unused5	0x00000400	This flag MUST be ignored on receipt.
BestBody	0x00002000	This flag MUST only be passed if the value of the **InputServerObject** field is a Message object. For more details, see section 3.2.5.8.1.1. **Best body** logic is specified in [MS-OXBBODY].<3>

SendOptions (1 byte): An 8-bit flag structure. This field defines the data representation parameters of the download operation.

The following table defines valid flags for the **SendOptions** field.

Flag name	Value	Description
Unicode	0x01	This flag indicates whether string properties are output in Unicode or in the **code page** set on the current connection. For details about the **Unicode** and **ForceUnicode** flags, and relationship between the two, see section 3.2.5.8.1.1. When used with the **RopSynchronizationConfigure** ROP (section 2.2.3.2.1.1.1), the value of this flag MUST match the value of the **Unicode** flag of the **SynchronizationFlags** field, as specified in section 2.2.3.2.1.1.1.

Flag name	Value	Description
UseCpid	0x02	This flag indicates support for code page property types, as specified in section 2.2.4.1.1.1. This flag MUST be set for server-to-client-to-server uploads only. For more details about server-to-client-to-server uploads, see section 3.3.4.2.1. If this flag is set, the **Unicode** flag MUST also be set.
ForUpload	0x03	This flag is the combination of the **Unicode** and **UseCpid** flags. This flag indicates that the client is requesting the FastTransfer stream for immediate upload to another destination server. This flag MUST be set for server-to-client-to-server uploads only. For more details about server-to-client-to-server uploads, see section 3.3.4.2.1. The ROP that uses this flag MUST be followed by the **RopTellVersion** ROP. For details about how this affects behaviors of servers and clients, see section 3.3.4.2.1.
RecoverMode	0x04	If this flag is set, it indicates that the client supports recovery mode. For more details about server behavior when this flag is set, see section 3.2.5.8.1.1. If this flag is not set, it indicates that the client does not support recovery mode. This flag MUST NOT be set when the **ForUpload** flag is set.
ForceUnicode	0x08	This flag indicates whether string properties are output in Unicode. For details about the **Unicode** and **ForceUnicode** flags, and relationship between the two, see section 3.2.5.8.1.1.
PartialItem	0x10	This flag MUST only be set for content synchronization download operations. This flag SHOULD<4> be set if the client supports partial item downloads.
Reserved1	0x20	This flag MUST be set to 0 when sent and MUST be ignored when received.
Reserved2	0x40	This flag MUST be set to 0 when sent and MUST be ignored when received.

PropertyTagCount (2 bytes): An unsigned 16-bit integer. This value specifies the number of structures in the **PropertyTags** field. If the value of the **PropertyTagCount** field is set to 0x0000, the **PropertyTags** field contains an empty array.

PropertyTags (variable): An array of **PropertyTag** structures ([MS-OXCDATA] section 2.9). Specifies properties and subobjects, as specified in section 2.2.1.7, to be excluded when copying a messaging object pointed to by the **InputServerObject** field. This field does not determine what properties and subobjects the server copies for descendant subobjects of the **InputServerObject** field. For more details about the effect of property and subobject filters on download operations, see section 3.2.5.10.

Remarks:

If, for example, the **InputServerObject** field contains a folder that was opened to show soft deleted messages (such as the **Deleted Items folder**), the scope of an operation that this ROP initiates only includes soft deleted messages. Otherwise, only normal, nondeleted messages are included. This applies to all valid values of the **Level** field.

2.2.3.1.1.1.2 RopFastTransferSourceCopyTo ROP Response Buffer

The following descriptions define valid fields for the response buffer of the **RopFastTransferSourceCopyTo** ROP ([MS-OXCROPS] section 2.2.12.6).

ReturnValue (4 bytes): An unsigned 32-bit integer. This value represents the ROP execution status.

OutputServerObject: The value of this field MUST be the FastTransfer download context. This value MUST be present if and only if the value of the **ReturnValue** field equals **Success** (0x00000000).

2.2.3.1.1.2 RopFastTransferSourceCopyProperties ROP

The **RopFastTransferSourceCopyProperties** ROP ([MS-OXCROPS] section 2.2.12.7) initializes a FastTransfer operation to download content from a specified messaging object and its descendant subobjects.

This ROP is very similar to the **RopFastTransferSourceCopyTo** ROP (section 2.2.3.1.1.1), with the following exceptions:

- The **PropertyTags** field specifies a list of properties and subobjects to include, as opposed to exclude.
- Best body logic, as indicated by the **BestBody** flag of the **CopyFlag** field specified in section 2.2.3.1.1.1.1, SHOULD NOT be used when copying messages.

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

For details about client behaviors related to this ROP, see sections 3.3.4.1 and 3.3.4.2.1. For details about server behaviors related to this ROP, see sections 3.2.5.8.1 and 3.2.5.8.1.2.

2.2.3.1.1.2.1 RopFastTransferSourceCopyProperties ROP Request Buffer

The following descriptions define valid fields for the request buffer of the **RopFastTransferSourceCopyProperties** ROP ([MS-OXCROPS] section 2.2.12.7).

InputServerObject: This value of this field MUST be either an Attachment object, Message object, or Folder object.

Level (1 byte): An unsigned 8-bit integer. This value MUST be set to 0x00 to copy descendant subobjects by using the property list specified by the **PropertyTags** field. This value MUST be set to a nonzero value to exclude all descendant subobjects from being copied. A nonzero value can only be passed when the value of the **InputServerObject** field is a Message object or Folder object. The value of the **Level** field MUST be ignored and treated as if it is set to 0x00 if the value of the **InputServerObject** field is an Attachment object.

CopyFlags (1 byte): An 8-bit flag structure. This field defines the parameters of the FastTransfer download operation.

The following table defines valid flags for the **CopyFlags** field.

Flag name	Value	Description
Move	0x01	This flag MUST only be set if the **InputServerObject** field is a Folder object or a Message object. If this flag is set, the FastTransfer operation is being configured as a logical part of a larger object move operation, as opposed to a copy operation, and the client will issue further operations such as deleting the moved messages from the source. If this flag is not set, the FastTransfer operation is not being configured as a logical part of a larger object move operation.
Unused1	0x02	This flag MUST be ignored on receipt.
Unused2	0x04	This flag MUST be ignored on receipt.

Flag name	Value	Description
Unused3	0x08	This flag MUST be ignored on receipt.

SendOptions (1 byte): An 8-bit flag structure. The possible values for this structure are defined in section 2.2.3.1.1.1.1.

PropertyTagCount (2 bytes): An unsigned 16-bit integer. This value specifies the number of structures in the **PropertyTags** field. This value MUST NOT be set to 0x0000.

PropertyTags (variable): An array of **PropertyTag** structures ([MS-OXCDATA] section 2.9). This array specifies the properties and subobjects, as specified in section 2.2.1.7, to copy from the messaging object pointed to by the **InputServerObject** field. Note that this field MUST NOT be considered when determining what properties and subobjects to copy for descendant subobjects of the **InputServerObject** field. For more details about the effect of property and subobject filters on download operations, see section 3.2.5.10.

2.2.3.1.1.2.2 RopFastTransferSourceCopyProperties ROP Response Buffer

The following descriptions define valid fields for the response buffer of the **RopFastTransferSourceCopyProperties** ROP ([MS-OXCROPS] section 2.2.12.7).

ReturnValue (4 bytes): An unsigned 32-bit integer. This value represents the ROP execution status.

OutputServerObject: The value of this field MUST be the FastTransfer download context. This value MUST be present if and only if the value of the **ReturnValue** field equals **Success** (0x00000000).

2.2.3.1.1.3 RopFastTransferSourceCopyMessages ROP

The **RopFastTransferSourceCopyMessages** ROP ([MS-OXCROPS] section 2.2.12.5) initializes a FastTransfer operation on a folder for downloading content and descendant subobjects of messages identified by a set of **MID** structures ([MS-OXCDATA] section 2.2.1.2).

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

For more details about client behaviors related to this ROP, see sections 3.3.4.1 and section 3.3.4.2.1. For more details about server behaviors related to this ROP, see sections 3.2.5.8.1 and 3.2.5.8.1.3.

2.2.3.1.1.3.1 RopFastTransferSourceCopyMessages ROP Request Buffer

The following descriptions define valid fields for the request buffer of the **RopFastTransferSourceCopyMessages** ROP ([MS-OXCROPS] section 2.2.12.5).

InputServerObject: The value of this field MUST be a Folder object.

MessageIdCount (2 bytes): An unsigned 16-bit integer. This value specifies the number of identifiers in the **MessageIds** field. The value of this field MUST be greater than zero.

MessageIds (variable): An array of 64-bit identifiers. This list specifies the **MID** structures ([MS-OXCDATA] section 2.2.1.2) of the messages to be copied. Messages MUST be contained by a folder identified by the **InputServerObject** field.

CopyFlags (1 byte): An 8-bit flag structure. This field defines the parameters of the FastTransfer download operation.

The following table defines valid bit flags for the **CopyFlags** field.

Flag name	Value	Description
Move	0x01	This bit flag MUST only be set if the value of the **InputServerObject** field is a Folder object. If this bit flag is set, the FastTransfer operation is being configured as a logical part of a larger object move operation, as opposed to a copy operation, and the client will issue further operations such as deleting the moved messages from the source. If this bit flag is not set, the FastTransfer operation is not being configured as a logical part of a larger object move operation. For more details, see section 3.2.5.8.1.3.
Unused1	0x02	This bit flag MUST be ignored on receipt.
Unused2	0x04	This bit flag MUST be ignored on receipt.
Unused3	0x08	The client MUST NOT set this flag.
BestBody	0x10	This bit flag MAY<5> identify whether the output **message body** is in the original format or compressed **RTF** format. For more details, see section 3.2.5.8.1.3.
SendEntryId	0x20	This bit flag indicates whether message change information is included in the FastTransfer stream. If this bit flag is set, the **PidTagSourceKey** (section 2.2.1.2.5), **PidTagChangeKey** (section 2.2.1.2.7), **PidTagLastModificationTime** ([MS-OXCMSG] section 2.2.2.2) and **PidTagPredecessorChangeList** (section 2.2.1.2.8) properties are included in the FastTransfer stream. In addition, the value of the **PidTagEntryId** property ([MS-OXCPERM] section 2.2.4) is assigned to the **PidTagOriginalEntryId** property (section 2.2.1.2.9). If this bit flag is not set, the **PidTagSourceKey**, **PidTagChangeKey**, **PidTagLastModificationTime**, **PidTagPredecessorChangeList**, and **PidTagOriginalEntryId** properties are not included in the FastTransfer stream. For more details, see section 3.2.5.8.1.3.

SendOptions (1 byte): An 8-bit flag structure. The possible values for this structure are defined in section 2.2.3.1.1.1.1.

2.2.3.1.1.3.2 RopFastTransferSourceCopyMessages ROP Response Buffer

The following descriptions define valid fields for the response buffer of the **RopFastTransferSourceCopyMessages** ROP ([MS-OXCROPS] section 2.2.12.5).

ReturnValue (4 bytes): An unsigned 32-bit integer. This value represents the ROP execution status.

OutputServerObject: The value of this field MUST be the FastTransfer download context. This value MUST be present if and only if the value of the **ReturnValue** field equals **Success** (0x00000000).

2.2.3.1.1.4 RopFastTransferSourceCopyFolder ROP

The **RopFastTransferSourceCopyFolder** ROP ([MS-OXCROPS] section 2.2.12.4) initializes a FastTransfer operation to download properties and descendant subobjects for a specified folder.

This ROP is very similar to **RopFastTransferSourceCopyTo**, with the following exceptions:

- The value of the **InputServerObject** field is limited to a Folder object.
- The FastTransfer stream produced by an operation configured with this ROP wraps folder properties and subobjects with the **topFolder** element (as specified in section 2.2.4.4).
- All properties and contained messages are copied.
- The **CopySubfolders** flag of the **CopyFlag** field indicates whether subfolders are to be copied.

- Best body logic, as indicated by the **BestBody** flag of the **CopyFlag** field specified in section 2.2.3.1.1.1.1, SHOULD NOT be used when copying messages.

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

For details about client behaviors related to this ROP, see sections 3.3.4.1 and section 3.3.4.2.1. For details about server behaviors related to this ROP, see sections 3.2.5.8.1 and 3.2.5.8.1.4.

2.2.3.1.1.4.1 RopFastTransferSourceCopyFolder ROP Request Buffer

The following descriptions define valid fields for the request buffer of the **RopFastTransferSourceCopyFolder** ROP ([MS-OXCROPS] section 2.2.12.4).

InputServerObject: The value of this field MUST be a Folder object.

CopyFlags (1 byte): An 8-bit flag structure. This field defines the parameters of the FastTransfer download operation.

The following table defines valid flags for the **CopyFlags** field.

Flag name	Value	Description
Move	0x01	This flag SHOULD<6> be ignored on receipt, or MAY<7> be set on a download operation to indicate the following: ▪ The FastTransfer operation is being configured as a logical part of a larger object move operation, as opposed to a copy operation, and the client will issue further operations such as deleting the moved messages from the source. ▪ The server does not output any objects in a FastTransfer stream that the client does not have permissions to delete. When the **Move** flag is not ignored, if the **Move** flag is not set, the FastTransfer operation is not being configured as a logical part of a larger object move operation.
Unused1	0x02	The client MUST NOT set this flag.
Unused2	0x04	The client MUST NOT set this flag.
Unused3	0x08	The client MUST NOT set this flag.
CopySubfolders	0x10	This flag identifies whether the subfolders of the folder specified in the **InputServerObject** field are recursively included in the scope. For more details, see section 3.2.5.8.1.4.

SendOptions (1 byte): An 8-bit flag structure. The possible values for this structure are defined in section 2.2.3.1.1.1.

2.2.3.1.1.4.2 RopFastTransferSourceCopyFolder ROP Response Buffer

The following descriptions define valid fields for the response buffer of the **RopFastTransferSourceCopyFolder** ROP ([MS-OXCROPS] section 2.2.12.4).

ReturnValue (4 bytes): An unsigned 32-bit integer. This value represents the ROP execution status.

OutputServerObject: The value of this field MUST be the FastTransfer download context. This value MUST be present if and only if the value of the **ReturnValue** field equals **Success** (0x00000000).

2.2.3.1.1.5 RopFastTransferSourceGetBuffer ROP

The **RopFastTransferSourceGetBuffer** ROP ([MS-OXCROPS] section 2.2.12.3) downloads the next portion of a FastTransfer stream that is produced by a previously configured download operation.

The **RopFastTransferSourceGetBuffer** ROP supports packed buffers, as specified in [MS-OXCRPC] section 3.1.4.2.1.2.

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

For more details about client behaviors related to this ROP, see sections 3.3.4.1 and 3.3.5.7.1. For more details about server behaviors related to this ROP, see sections 3.2.5.8.1 and 3.2.5.8.1.5.

2.2.3.1.1.5.1 RopFastTransferSourceGetBuffer ROP Request Buffer

The following descriptions define valid fields for the request buffer of the **RopFastTransferSourceGetBuffer** ROP ([MS-OXCROPS] section 2.2.12.3).

InputServerObject: The value of this field MUST be the download context.

BufferSize (2 bytes): An unsigned 16-bit integer. This field specifies the maximum amount of data (in bytes) to be output in by the **TransferBuffer** field. For more details, see sections 3.3.5.7.1 and 3.2.5.8.1.5.

MaximumBufferSize (2 bytes, optional): An unsigned 16-bit integer that specifies the maximum size limit when the server determines the buffer size. This value MUST be present if and only if the value of the **BufferSize** field is set to a sentinel value of 0xBABE.

2.2.3.1.1.5.2 RopFastTransferSourceGetBuffer ROP Response Buffer

The following descriptions define valid fields for the response buffer of the **RopFastTransferSourceGetBuffer** ROP ([MS-OXCROPS] section 2.2.12.3).

ReturnValue (4 bytes): An unsigned 32-bit integer. This value represents the ROP execution status. For more details, see sections 3.3.5.7.1 and 3.2.5.8.1.5.

TransferStatus (2 bytes): A 16-bit enumeration. This field represents the status of the download operation after producing data for the **TransferBuffer** field.

The following table defines valid values for the **TransferStatus** field.

Value	Bit	Description
Error	0x0000	The download stopped because a nonrecoverable error has occurred when producing a FastTransfer stream. The **ReturnValue** field of the ROP response buffer contains a code for that error.
Partial	0x0001	The FastTransfer stream was split, and more data is available. **TransferBuffer** contains incomplete data. For details about where to split FastTransfer streams, see section 2.2.4.1.
NoRoom	0x0002	The FastTransfer stream was split, more data is available, and the value of the **TransferBuffer** field contains incomplete data. The server MAY<8> set this value.
Done	0x0003	This was the last portion of the FastTransfer stream.

InProgressCount (2 bytes): An unsigned 16-bit integer. The number of steps that have already been completed in the current operation. This value is only usable for progress information display.

TotalStepCount (2 bytes): An unsigned 16-bit integer that contains the approximate total number of steps to be completed in the current operation. This value MAY<9> contain accurate information and is only usable for progress information display.

Reserved (1 byte): The value of this field MUST be set to 0x00 when sending and ignored on receipt.

TransferBufferSize (2 bytes): An unsigned 16-bit integer. This value specifies the size of the **TransferBuffer** field.

TransferBuffer (variable, optional): An array of bytes that contains the next portion of a FastTransfer stream. The syntax of the FastTransfer stream is specified in section 2.2.4. This field SHOULD only exist if the value of the **ReturnValue** field is **Success** (0x00000000).

BackoffTime (4 bytes, optional): An unsigned 32-bit integer that contains the time, in milliseconds, that a client waits before retrying the ROP. This field MUST be present if and only if the value of the **ReturnValue** field is 0x00000480, as specified in [MS-OXCDATA] section 2.4.

2.2.3.1.1.6 RopTellVersion ROP

The **RopTellVersion** ROP ([MS-OXCROPS] section 2.2.12.8) is used to provide the version of one server to another server that is participating in the server-to-client-to-server upload, as specified in section 3.3.4.2.1.

For more details about client behaviors related to this ROP, see sections 3.3.4.1 and 3.3.5.7.2. For more details about server behaviors related to this ROP, see sections 3.2.5.8.1 and 3.2.5.8.1.6.

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

2.2.3.1.1.6.1 RopTellVersion ROP Request Buffer

The following descriptions define valid fields for the request buffer of the **RopTellVersion** ROP ([MS-OXCROPS] section 2.2.12.8).

Version (6 bytes): An array of three unsigned 16-bit integers. This array contains the version information for another server that is participating in the server-to-client-to-server upload. The format of this structure is the same as that specified in [MS-OXCRPC] section 3.1.4.1.3.1.

2.2.3.1.1.6.2 RopTellVersion ROP Response Buffer

The following descriptions define valid fields for the response buffer of the **RopTellVersion** ROP ([MS-OXCROPS] section 2.2.12.8).

ReturnValue (4 bytes): An unsigned 32-bit integer. This value represents the ROP execution status.

2.2.3.1.2 Upload

2.2.3.1.2.1 RopFastTransferDestinationConfigure ROP

The **RopFastTransferDestinationConfigure** ROP ([MS-OXCROPS] section 2.2.12.1) initializes a FastTransfer operation for uploading content encoded in a client-provided FastTransfer stream into a mailbox.

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

For more details about client behaviors related to this ROP, see sections 3.3.4.2 and section 3.3.4.2.1. For more details about server behaviors related to this ROP, see section 3.2.5.8.2.1.

2.2.3.1.2.1.1 RopFastTransferDestinationConfigure ROP Request Buffer

The following descriptions define valid fields for the request buffer of the **RopFastTransferDestinationConfigure** ROP ([MS-OXCROPS] section 2.2.12.1).

InputServerObject: The value of this field MUST be either an Attachment object, Message object, or Folder object.

SourceOperation (1 byte): An 8-bit enumeration. This enumeration is used to specify the type of data in a FastTransfer stream that is uploaded by using the **RopFastTransferDestinationPutBuffer** ROP (section 2.2.3.1.2.2) on the FastTransfer upload context that is returned in the **OutputServerObject** field.

The following table defines the **SourceOperation** enumeration values and the associated root elements in the FastTransfer stream.

SourceOperation enumeration value	Root element in FastTransfer stream	Conditions
CopyTo **CopyProperties**	**folderContent** element	The value of the **InputServerObject** field is a Folder object.
	messageContent element	The value of the **InputServerObject** field is a Message object.
	attachmentContent element	The value of the **InputServerObject** field is an Attachment object.
CopyMessages	**messageList** element	Always.
CopyFolder	**topFolder** element	Always.

The following table defines the **SourceOperation** ordinal values.

SourceOperation enumeration value	Ordinal value	Corresponding ROP of the FastTransfer download*
CopyTo	0x01	**RopFastTransferSourceCopyTo** ROP (section 2.2.3.1.1.1)
CopyProperties	0x02	**RopFastTransferSourceCopyProperties** ROP (section 2.2.3.1.1.2)
CopyMessages	0x03	**RopFastTransferSourceCopyMessages** ROP (section 2.2.3.1.1.3)
CopyFolder	0x04	**RopFastTransferSourceCopyFolder** ROP (section 2.2.3.1.1.4)

*If the FastTransfer stream to be uploaded was produced by a FastTransfer download operation, the value of the **SourceOperation** field MUST correspond to the **RopFastTransferSourceCopy*** ROP that was used to configure the download operation.

CopyFlags (1 byte): An 8-bit flag structure. This field defines the parameters of the FastTransfer upload operation.

The following table defines valid flags for the **CopyFlags** field.

Flag name	Value	Description
Move	0x01	This flag MUST only be set if the value of the **InputServerObject** field is a Folder object or Message object. If this flag is set, the FastTransfer operation is being configured as a logical part of a larger object move operation, as opposed to a copy operation, and the client will issue further operations such as deleting the moved messages from the source. If this flag is not set, the FastTransfer operation is not being configured as a logical part of a larger object move operation.

2.2.3.1.2.1.2 RopFastTransferDestinationConfigure ROP Response Buffer

The following descriptions define valid fields for the response buffer of the **RopFastTransferDestinationConfigure** ROP ([MS-OXCROPS] section 2.2.12.1).

ReturnValue (4 bytes): An unsigned 32-bit integer. This value represents the ROP execution status.

OutputServerObject: The value of this field MUST be the FastTransfer upload context. This field MUST be present if and only if the value of the **ReturnValue** field equals **Success** (0x00000000).

2.2.3.1.2.2 RopFastTransferDestinationPutBuffer ROP

The **RopFastTransferDestinationPutBuffer** ROP ([MS-OXCROPS] section 2.2.12.2) uploads the next portion of an input FastTransfer stream for a previously configured FastTransfer upload operation.

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

2.2.3.1.2.2.1 RopFastTransferDestinationPutBuffer ROP Request Buffer

The following descriptions define valid fields for the request buffer of the **RopFastTransferDestinationPutBuffer** ROP ([MS-OXCROPS] section 2.2.12.2).

InputServerObject: The value of this field MUST be the FastTransfer upload context.

TransferDataSize (2 bytes): An unsigned 16-bit integer. This value specifies the size of the **TransferData** field. The value MUST NOT be set to 0x0000. The maximum data size is limited to the available space allowed by the underlying transport used by the **EcDoRpcExt2** method ([MS-OXCRPC] section 3.1.4.2) or by the **Execute** request type<10> ([MS-OXCMAPIHTTP] section 2.2.4.2).

TransferData (variable): An array of bytes. This array contains the data to be uploaded to the destination **FastTransfer** object and contains the next portion of a FastTransfer stream. The syntax of the FastTransfer stream is specified in section 2.2.4.

2.2.3.1.2.2.2 RopFastTransferDestinationPutBuffer ROP Response Buffer

The following descriptions define valid fields for the response buffer of the **RopFastTransferDestinationPutBuffer** ROP ([MS-OXCROPS] section 2.2.12.2).

ReturnValue (4 bytes): An unsigned 32-bit integer. This value represents the ROP execution status.

TransferStatus (2 bytes): A 16-bit enumeration. Clients MUST ignore the value of this field.

InProgressCount (2 bytes): An unsigned 16-bit integer. This value specifies the number of steps that have been completed in the current operation. The server SHOULD set this field to 0x0000. This field is usable only for progress information display.

TotalStepCount (2 bytes): An unsigned 16-bit integer. This field contains the approximate total number of steps to be completed in the current operation. This field is used only to display progress information. However, the value is implementation-specific<11> and not required for interoperability.

Reserved (1 byte): The field MUST be set to 0x00 when sending, and MUST be ignored on receipt.

BufferUsedSize (2 bytes): An unsigned 16-bit integer. The value SHOULD<12> be the same size as the value of **TransferDataSize** field if the value of the **ReturnValue** field is equal to **Success** (0x00000000), otherwise it can be equal to or less than the value of the **TransferDataSize** field on failure when the value of the **ReturnValue** field is not equal to **Success** (0x00000000).

2.2.3.2 Incremental Change Synchronization

2.2.3.2.1 Download

2.2.3.2.1.1 RopSynchronizationConfigure ROP

The **RopSynchronizationConfigure** ROP ([MS-OXCROPS] section 2.2.13.1) is used to define the synchronization scope and parameters of the synchronization download operation.

The synchronization scope determines the boundaries of a synchronization operation, and is defined by the following:

- The type of objects considered for synchronization (folders for hierarchy synchronization operations, and messages for content synchronization operations).
- The folder that contains these objects as children (contents) or descendants (hierarchy).
- The restriction on messages within that folder (contents).

For more details about determining the synchronization scope, see section 3.3.5.5.

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

For more details about client behaviors related to this ROP, see sections 3.3.4.3.2 and 3.3.5.8.1. For more details about server behaviors related to this ROP, see section 3.2.5.9.1.1.

2.2.3.2.1.1.1 RopSynchronizationConfigure ROP Request Buffer

The following descriptions define valid fields for the request buffer of the **RopSynchronizationConfigure** ROP ([MS-OXCROPS] section 2.2.13.1).

InputServerObject: The value of this field MUST be a Folder object that contributes to the synchronization scope.

SynchronizationType (1 byte): An 8-bit enumeration that defines the type of synchronization requested: contents or hierarchy. This field contributes to the synchronization scope.

The following table defines valid values for the **SynchronizationType** field.

Value	Bit	Description
Contents	0x01	Indicates a content synchronization operation.

Value	Bit	Description
Hierarchy	0x02	Indicates a hierarchy synchronization operation.

SendOptions (1 byte): An 8-bit enumeration that identifies options for sending the data. For details about the possible values for this enumeration, see section 2.2.3.1.1.1.1.

SynchronizationFlags (2 bytes): A 16-bit flag structure that defines the parameters of the synchronization operation.

The following table defines valid flags for the **SynchronizationFlags** field.

Flag name	Value	Description
Unicode	0x0001	Indicates whether the client supports Unicode. For more details, see section 3.2.5.9.1.1. This flag MUST match the value of the Unicode flag from the **SendOptions** field.
NoDeletions	0x0002	Indicates how the server downloads information about item deletions. For more details, see section 3.2.5.9.1.1. The client MAY implement this flag.
IgnoreNoLongerInScope	0x0004	Indicates whether the server downloads information about messages that went out of scope. For more details, see section 3.2.5.9.1.1. This flag MUST only be set for a content synchronization download operation. The client MAY implement this flag.
ReadState	0x0008	Indicates whether the server downloads information about changes to the read state of messages. For more details, see section 3.2.5.9.1.1. This flag MUST only be set for a content synchronization download operation.
FAI	0x0010	Indicates whether the server downloads information about changes to FAI messages. For more details, see section 3.2.5.9.1.1. This flag MUST only be set for a content synchronization download operation.
Normal	0x0020	Indicates whether the server downloads information about changes to normal messages. For more details, see section 3.2.5.9.1.1. This flag MUST only be passed for a content synchronization download operation.
OnlySpecifiedProperties	0x0080	Indicates whether the server limits or excludes properties and subobjects output to the properties listed in **PropertyTags**. For more details, see section 3.2.5.9.1.1. This flag MUST only be passed for a content synchronization download operation.
NoForeignIdentifiers	0x0100	Identifies whether the server ignores any persisted values for the **PidTagSourceKey** property (section 2.2.1.2.5) and **PidTagParentSourceKey** property (section 2.2.1.2.6) when producing output for folder and message changes. For more details, see section 3.2.5.9.1.1. Clients SHOULD set this flag. For more details about possible issues if this flag is not set, see section 3.3.5.2.3.
Reserved	0x1000	This flag MUST be set to 0 when sending.

Flag name	Value	Description
BestBody	0x2000	Identifies whether the server outputs message bodies in their original format or in RTF.<13> For more details, see section 3.2.5.9.1.1. This flag MUST only be passed a content synchronization download operation.
IgnoreSpecifiedOnFAI	0x4000	Indicates whether the server outputs properties and subobjects of FAI messages. For more details, see section 3.2.5.9.1.1. This flag MUST only be passed for a content synchronization download operation.
Progress	0x8000	Indicates whether the server injects progress information into the output FastTransfer stream. For more details, see section 3.2.5.9.1.1. This flag MUST only be passed for a content synchronization download operation. This flag is in addition to the means of progress reporting available through the **RopFastTransferSourceGetBuffer** ROP results.

RestrictionDataSize (2 bytes): An unsigned 16-bit integer that specifies the length of the **RestrictionData** field. This value MUST be set to 0x0000 if the **SynchronizationType** field is set to **Hierarchy** ("0x02").

RestrictionData (variable): The variable-length restriction structure that is used to limit the data to be synchronized. This value contributes to the synchronization scope. This field is used in content synchronization operations only. This field is not applicable if the value of the **SynchronizationType** field is set to **Hierarchy** (0x02) as the **RestrictionDataSize** field MUST be set to 0x0000. For more details about restrictions, see [MS-OXCDATA] section 2.12.

SynchronizationExtraFlags (4 bytes): A 32-bit flag structure that defines additional parameters of the synchronization operation.

The following table defines valid bit flags for the **SynchronizationExtraFlags** field.

Flag name	Value	Description
Eid	0x00000001	Indicates whether the server includes the **PidTagFolderId** (section 2.2.1.2.2) or **PidTagMid** (section 2.2.1.2.1) properties in the folder change or message change header. For more details, see section 3.2.5.9.1.1.
MessageSize	0x00000002	Indicates whether the server includes the **PidTagMessageSize** property (section 2.2.1.6) in the message change header. For more details, see section 3.2.5.9.1.1. This flag MUST only be set for a content synchronization download operation.
CN	0x00000004	Indicates whether the server includes the **PidTagChangeNumber** property (section 2.2.1.2.3) in the message change header. For more details, see section 3.2.5.9.1.1. This flag MUST only be set for a content synchronization download operation.
OrderByDeliveryTime	0x00000008	Indicates whether the server sorts messages by their delivery time. For more details, see section 3.2.5.9.1.1. This flag MUST only be set for a content synchronization download operation.

PropertyTagCount (2 bytes): An unsigned 16-bit integer that specifies the number of **PropertyTag** structures in the value of the **PropertyTags** field. The value of the **PropertyTagCount** field is set to 0x0000 if the value of the **PropertyTags** field is an empty array.

PropertyTags (variable): An array of **PropertyTag** structures that contains property tags for regular properties and properties that denote message subobjects, as specified in section 2.2.1.7, to exclude or include in the synchronization scope. The behavior of this field is dependent on the **OnlySpecifiedProperties** flag of the **SynchronizationFlags** field.

For more details about this field, see section 3.2.5.9.1.1.

2.2.3.2.1.1.2 RopSynchronizationConfigure ROP Response Buffer

The following descriptions define valid fields for the response buffer of the **RopSynchronizationConfigure** ROP ([MS-OXCROPS] section 2.2.13.1).

ReturnValue (4 bytes): An unsigned 32-bit integer. This value represents the status of the ROP execution.

OutputServerObject: This value MUST be the synchronization download context. This value MUST be present if and only if **ReturnValue** is **Success** (0x00000000).

2.2.3.2.2 Upload State

2.2.3.2.2.1 RopSynchronizationUploadStateStreamBegin ROP

The **RopSynchronizationUploadStateStreamBegin** ROP ([MS-OXCROPS] section 2.2.13.9) initiates the upload of an ICS state property into the **synchronization context**.

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

For more details about client behaviors related to this ROP, see sections 3.3.4.3.1 and 3.3.5.8.2. For more details about server behaviors related to this ROP, see section 3.2.5.9.2.1.

2.2.3.2.2.1.1 RopSynchronizationUploadStateStreamBegin ROP Request Buffer

The following descriptions define valid fields for the request buffer of the **RopSynchronizationUploadStateStreamBegin** ROP ([MS-OXCROPS] section 2.2.13.9).

InputServerObject: The value of this field MUST be a synchronization context.

StateProperty (4 bytes): A 32-bit **PropertyTag** structure. Valid input is restricted to the property tags of the ICS state properties: **MetaTagIdsetGiven** (section 2.2.1.1.1), **MetaTagCnsetSeen** (section 2.2.1.1.2), **MetaTagCnsetSeenFAI** (section 2.2.1.1.3), and **MetaTagCnsetRead** (section 2.2.1.1.4).

TransferBufferSize (4 bytes): An unsigned 32-bit integer. This value specifies the size of the stream to be uploaded by the **RopSynchronizationUploadStateStreamContinue** ROP.

2.2.3.2.2.1.2 RopSynchronizationUploadStateStreamBegin ROP Response Buffer

The following descriptions define valid fields for the response buffer of the **RopSynchronizationUploadStateStreamBegin** ROP ([MS-OXCROPS] section 2.2.13.9).

ReturnValue (4 bytes): An unsigned 32-bit integer. This value represents the ROP execution status.

2.2.3.2.2.2 RopSynchronizationUploadStateStreamContinue ROP

The **RopSynchronizationUploadStateStreamContinue** ROP ([MS-OXCROPS] section 2.2.13.10) continues to upload an ICS state property value into the synchronization context.

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

For more details about client behaviors related to this ROP, see sections 3.3.4.3.1 and 3.3.5.8.3. For more details about server behaviors related to this ROP, see section 3.2.5.9.2.2.

2.2.3.2.2.2.1 RopSynchronizationUploadStateStreamContinue ROP Request Buffer

The following descriptions define valid fields for the request buffer of the **RopSynchronizationUploadStateStreamContinue** ROP ([MS-OXCROPS] section 2.2.13.10).

InputServerObject: The value of this field MUST be a synchronization context.

StreamDataSize (4 bytes): An unsigned 32-bit integer. This value specifies the size of the **StreamData** field. The value of this field MUST NOT be set to 0x00000000.

StreamData (variable): This array contains the state stream data to be uploaded.

2.2.3.2.2.2.2 RopSynchronizationUploadStateStreamContinue ROP Response Buffer

The following descriptions define valid fields for the response buffer of the **RopSynchronizationUploadStateStreamContinue** ROP ([MS-OXCROPS] section 2.2.13.10).

ReturnValue (4 bytes): An unsigned 32-bit integer. This value represents the ROP execution status.

2.2.3.2.2.3 RopSynchronizationUploadStateStreamEnd ROP

The **RopSynchronizationUploadStateStreamEnd** ROP ([MS-OXCROPS] section 2.2.13.11) concludes the upload of an ICS state property value into the synchronization context.

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

For more details about client behaviors related to this ROP, see sections 3.3.4.3.1 and 3.3.5.8.4. For more details about server behaviors related to this ROP, see section 3.2.5.9.2.3.

2.2.3.2.2.3.1 RopSynchronizationUploadStateStreamEnd ROP Request Buffer

The following descriptions define valid fields for the request buffer of the **RopSynchronizationUploadStateStreamEnd** ROP ([MS-OXCROPS] section 2.2.13.11).

InputServerObject: The value of this field MUST be a synchronization context.

2.2.3.2.2.3.2 RopSynchronizationUploadStateStreamEnd ROP Response Buffer

The following descriptions define valid fields for the response buffer of the **RopSynchronizationUploadStateStreamEnd** ROP ([MS-OXCROPS] section 2.2.13.11).

ReturnValue (4 bytes): An unsigned 32-bit integer. This value represents the ROP execution status.

2.2.3.2.3 Download State

2.2.3.2.3.1 RopSynchronizationGetTransferState ROP

The **RopSynchronizationGetTransferState** ROP ([MS-OXCROPS] section 2.2.13.8) creates a FastTransfer download context for the **checkpoint ICS state** of the operation identified by the given synchronization download context or synchronization upload context at the current moment in time.

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

For more details about client behaviors related to this ROP, see sections 3.3.4.3.4 and 3.3.5.8.5. For more details about server behaviors related to this ROP, see section 3.2.5.9.3.1.

2.2.3.2.3.1.1 RopSynchronizationGetTransferState ROP Request Buffer

The following descriptions define valid fields for the request buffer of the **RopSynchronizationGetTransferState** ([MS-OXCROPS] section 2.2.13.8).

InputServerObject: The value of this field MUST be either a synchronization download context or synchronization upload context.

2.2.3.2.3.1.2 RopSynchronizationGetTransferState ROP Response Buffer

The following descriptions define valid fields for the response buffer of the **RopSynchronizationGetTransferState** ([MS-OXCROPS] section 2.2.13.8).

ReturnValue (4 bytes): An unsigned 32-bit integer. This value represents the ROP execution status.

OutputServerObject: The value of this field MUST be the FastTransfer download context for the ICS state. This value MUST be present if and only if the value of the **ReturnValue** field equals **Success** (0x00000000).

2.2.3.2.4 Upload

2.2.3.2.4.1 RopSynchronizationOpenCollector ROP

The **RopSynchronizationOpenCollector** ROP ([MS-OXCROPS] section 2.2.13.7) configures the synchronization upload operation and returns a handle to a synchronization upload context.

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

For more details about client behaviors related to this ROP, see sections 3.3.4.3.3 and 3.3.5.8.6. For more details about server behaviors related to this ROP, see section 3.2.5.9.4.1.

2.2.3.2.4.1.1 RopSynchronizationOpenCollector ROP Request Buffer

The following descriptions define valid fields for the request buffer of the **RopSynchronizationOpenCollector** ROP ([MS-OXCROPS] section 2.2.13.7).

InputServerObject: The value of this field MUST be a Folder object that contributed to the synchronization scope that corresponds to the initial ICS state to be uploaded, as specified in section 3.3.5.5.

IsContentsCollector (1 byte): An 8-bit **PtypBoolean** ([MS-OXCDATA] section 2.11.1) value. This value is 0x01 (nonzero) if a synchronization upload is requested for contents of folders, or 0x00 if a synchronization upload is requested for the hierarchy of the folder contents.

2.2.3.2.4.1.2 RopSynchronizationOpenCollector ROP Response Buffer

The following descriptions define valid fields for the response buffer of the **RopSynchronizationOpenCollector** ROP ([MS-OXCROPS] section 2.2.13.7).

ReturnValue (4 bytes): An unsigned 32-bit integer. This value represents the ROP execution status.

OutputServerObject: The value of this field MUST be the synchronization upload context. This value MUST be present if and only if the value of the **ReturnValue** field equals **Success** (0x00000000).

2.2.3.2.4.2 RopSynchronizationImportMessageChange ROP

The **RopSynchronizationImportMessageChange** ROP ([MS-OXCROPS] section 2.2.13.2) is used to import new messages or changes to existing messages into the server replica. When there are changes to existing messages, the entire changed message MUST be uploaded.

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

For more details about client behaviors related to this ROP, see sections 3.3.4.3.3 and 3.3.5.8.7. For more details about server behaviors related to this ROP, see section 3.2.5.9.4.2.

2.2.3.2.4.2.1 RopSynchronizationImportMessageChange ROP Request Buffer

The following descriptions define valid fields for the request buffer of the **RopSynchronizationImportMessageChange** ROP ([MS-OXCROPS] section 2.2.13.2).

InputServerObject: The value of this field MUST be the synchronization upload context configured for the collection of changes to content.

ImportFlag (1 byte): An 8-bit flag structure that defines the parameters of the import operation.

The following table defines valid flags for the **ImportFlag** field.

Flag name	Value	Description
Associated	0x10	If this flag is set, the message being imported is an FAI message. If this flag is not set, the message being imported is a normal message.
FailOnConflict	0x40	The server SHOULD**<14>** support this flag. If this flag is set, the server accepts conflicting versions of a particular message. If this flag is not set, the server does not accept conflicting versions of a particular message. For more details, see section 3.2.5.9.4.2.

PropertyValueCount (2 bytes): An unsigned 16-bit integer. This value specifies the number of structures in the **PropertyValues** field. This value MUST NOT be set to 0x0000.

PropertyValues (variable): An array of **TaggedPropertyValue** structures ([MS-OXCDATA] section 2.11.4). These values are used to specify extra properties on the message; these are properties that cannot be set by using the **RopSetProperties** ROP.

The following table lists the restrictions that exist for properties passed in the **PropertyValues** field.

Property	Restrictions	Comments
PidTagSourceKey (section 2.2.1.2.5)	Required Fixed position	A **GID** structure ([MS-OXCDATA] section 2.2.1.3) that identifies the message being uploaded in the local replica.

Property	Restrictions	Comments
PidTagLastModificationTime ([MS-OXPROPS] section 2.755)	Required Fixed position	None.
PidTagChangeKey (section 2.2.1.2.7)	Required Fixed position	An **XID** structure, as specified in section 2.2.2.2, that identifies a change to a message being uploaded in a local replica. For details about how clients can generate this value, see section 3.1.5.3.
PidTagPredecessorChangeList (section 2.2.1.2.8)	Required Fixed position	None.
< other properties >	*Prohibited*	None.

2.2.3.2.4.2.2 RopSynchronizationImportMessageChange ROP Response Buffer

The following descriptions define valid fields for the response buffer of the **RopSynchronizationImportMessageChange** ROP ([MS-OXCROPS] section 2.2.13.2).

ReturnValue (4 bytes): An unsigned 32-bit integer. This value represents the ROP execution status. For details about the common return values for **RopSynchronizationImport*** ROPs that require special processing, see section 3.3.4.3.3. The following table contains one additional return value.

Return value name	Value	Description
SyncConflict	0x80040802	A conflict has occurred and conflict resolution failed. No data was imported.

OutputServerObject: The value of this field MUST be the Message object into which the client will upload the rest of the message changes. This value MUST be present if and only if **ReturnValue** equals **Success** (0x00000000).

MessageId (8 bytes): A 64-bit identifier that specifies the **MID** structure ([MS-OXCDATA] section 2.2.1.2) of the message that was imported. This value MUST be set to 0x0000000000000000. This value MUST be present if and only if **ReturnValue** equals **Success** (0x00000000).

2.2.3.2.4.3 RopSynchronizationImportHierarchyChange ROP

The **RopSynchronizationImportHierarchyChange** ROP ([MS-OXCROPS] section 2.2.13.4) is used to import new folders, or changes to existing folders, into the server replica.

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

For more details about client behaviors related to this ROP, see sections 3.3.4.3.3 and 3.3.5.8.8. For more details about server behaviors related to this ROP, see section 3.2.5.9.4.3.

2.2.3.2.4.3.1 RopSynchronizationImportHierarchyChange ROP Request Buffer

The following descriptions define valid fields for the request buffer of the **RopSynchronizationImportHierarchyChange** ROP ([MS-OXCROPS] section 2.2.13.4).

InputServerObject: The value of this field MUST be the synchronization upload context configured to collect changes to the hierarchy.

HierarchyValueCount (2 bytes): An unsigned 16-bit integer. This value specifies the number of structures in the **HierarchyValues** field. This value MUST NOT be set to 0x0000.

HierarchyValues (variable): An array of **TaggedPropertyValue** structures ([MS-OXCDATA] section 2.11.4). These values are used to specify folder hierarchy properties, which determine the location of the folder within the hierarchy. The following table lists the restrictions that exist on the array of values of the **HierarchyValues** field.

Property	Restrictions	Comments
PidTagParentSourceKey (section 2.2.1.2.6)	Required Fixed position	SHOULD<15> be zero-length to identify a folder for which a synchronization upload context was opened.
PidTagSourceKey (section 2.2.1.2.5)	Required Fixed position	A **GID** structure ([MS-OXCDATA] section 2.2.1.3) that identifies the folder being uploaded in the local replica.
PidTagLastModificationTime ([MS-OXPROPS] section 2.755)	Required Fixed position	None.
PidTagChangeKey (section 2.2.1.2.7)	Required Fixed position	An **XID** structure, as specified in section 2.2.2.2, that identifies the change being uploaded in the local replica. For details about how clients can generate the **PidTagChangeKey** value, see section 3.1.5.3.
PidTagPredecessorChangeList (section 2.2.1.2.8)	Required Fixed position	None.
PidTagDisplayName ([MS-OXCFOLD] section 2.2.2.2.2.5)	Required Fixed position	This value MUST be a nonempty string.
< other properties >	*Prohibited*	None.

PropertyValueCount (2 bytes): An unsigned 16-bit integer. This value specifies the number of structures in the **PropertyValues** field. The value of this field MUST NOT be set to 0x0000.

PropertyValues (variable): An array of **TaggedPropertyValue** structures ([MS-OXCDATA] section 2.11.1) that contains the changed folder properties.

2.2.3.2.4.3.2 RopSynchronizationImportHierarchyChange ROP Response Buffer

The following descriptions define valid fields for the response buffer of the **RopSynchronizationImportHierarchyChange** ROP ([MS-OXCROPS] section 2.2.13.4).

ReturnValue (4 bytes): An unsigned 32-bit integer. This value represents the ROP execution status. For details about common return values of the **RopSynchronizationImport*** ROPs that require special processing, see section 3.3.4.3.3.

FolderId (8 bytes): A 64-bit identifier that contains the **Folder ID** structure ([MS-OXCDATA] section 2.2.1.1) of the folder that was imported. This value of this field MUST be set to 0x0000000000000000. This value MUST be present if and only if the value of the **ReturnValue** field equals **Success** (0x00000000).

2.2.3.2.4.4 RopSynchronizationImportMessageMove ROP

The **RopSynchronizationImportMessageMove** ROP ([MS-OXCROPS] section 2.2.13.6) imports information about moving a message between two existing folders within the same mailbox.

To move folders within a mailbox, use the **RopSynchronizationImportHierarchyChange** ROP.

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

For more details about client behaviors related to this ROP, see sections 3.3.4.3.3 and 3.3.5.8.9. For more details about server behaviors related to this ROP, see section 3.2.5.9.4.4.

2.2.3.2.4.4.1 RopSynchronizationImportMessageMove ROP Request Buffer

The following descriptions define valid fields for the request buffer of the **RopSynchronizationImportMessageMove** ROP ([MS-OXCROPS] section 2.2.13.6).

InputServerObject: The value of this field MUST be the synchronization upload context configured for collecting changes to the contents of the message move destination folder.

SourceFolderIdSize (4 bytes): An unsigned 32-bit integer. This value specifies the size of the **SourceFolderId** field. This value of this field MUST NOT be set to 0x00000000.

SourceFolderId (variable): An array of bytes. This value contains a serialized representation of the **GID** structure ([MS-OXCDATA] section 2.2.1.3) that represents the **PidTagSourceKey** property (section 2.2.1.2.5) value of the source folder. The source folder MUST be in the same mailbox as the destination folder specified in **InputServerObject**.

SourceMessageIdSize (4 bytes): An unsigned 32-bit integer. This value specifies the size of the **SourceMessageId** field. This value of this field MUST NOT be set to 0x00000000.

SourceMessageId (variable): An array of bytes. This value contains a serialized representation of the **GID** structure that represents the **PidTagSourceKey** property of the message in the source folder, identified by **SourceFolderId** field.

PredecessorChangeListSize (4 bytes): An unsigned 32-bit integer. This value specifies the size of the **PredecessorChangeList** field. This value of this field MUST NOT be set to 0x00000000.

PredecessorChangeList (variable): An array of bytes. This value contains a serialized representation of the **PredecessorChangeList** structure, as specified in section 2.2.2.3, in the local replica of the message being moved.

DestinationMessageIdSize (4 bytes): An unsigned 32-bit integer. This value specifies the size of the **DestinationMessageId** field. This value of this field MUST NOT be set to 0x00000000.

DestinationMessageId (variable): An array of bytes. This value contains a serialized representation of the **GID** structure that represents the **PidTagSourceKey** property of the message in the destination folder. For details about why the value of the **DestinationMessageId** field is different from the value of the **SourceMessageId** field, see section 3.1.5.3.

ChangeNumberSize (4 bytes): An unsigned 32-bit integer. This value specifies the size of the **ChangeNumber** field. This value of this field MUST NOT be set to 0x00000000.

ChangeNumber (variable): An array of bytes. This value contains a serialized representation of an **XID** structure, as specified in section 2.2.2.2, that represents the **PidTagChangeKey** property (section 2.2.1.2.7) of the message in the destination folder.

2.2.3.2.4.4.2 RopSynchronizationImportMessageMove ROP Response Buffer

The following descriptions define valid fields for the response buffer of the **RopSynchronizationImportMessageMove** ROP ([MS-OXCROPS] section 2.2.13.6).

Return value (4 bytes): An unsigned 32-bit integer. This value represents the ROP execution status. For details about the common return values of the **RopSynchronizationImport*** ROPs that require special processing, see section 3.3.4.3.3. The following table contains one additional return value.

Return value name	Value	Description
NewerClientChange	0x00040821	The ROP succeeded, but the server replica had an older version of a message than the local replica. The values of the **ChangeNumber** and **PredecessorChangeList** fields, specified in section 2.2.3.2.4.4.1, were not applied to the destination message.

The complete list of error codes is specified in [MS-OXCDATA] section 2.4.

MessageId (8 bytes): A 64-bit identifier. The **MID** structure ([MS-OXCDATA] section 2.2.1.2) of the moved message in a destination folder. This value MUST be set to 0x0000000000000000. This value MUST be present if and only if **ReturnValue** equals **Success** (0x00000000).

2.2.3.2.4.5 RopSynchronizationImportDeletes ROP

The **RopSynchronizationImportDeletes** ROP ([MS-OXCROPS] section 2.2.13.5) imports deletions of messages or folders into the server replica.

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

For more details about client behaviors related to this ROP, see sections 3.3.4.3.3 and 3.3.5.8.10. For more details about server behaviors related to this ROP, see section 3.2.5.9.4.5.

2.2.3.2.4.5.1 RopSynchronizationImportDeletes ROP Request Buffer

The following descriptions define valid fields for the request buffer of the **RopSynchronizationImportDeletes** ROP ([MS-OXCROPS] section 2.2.13.5).

InputServerObject: The value of this field MUST be the synchronization upload context. The type of synchronization upload context MUST correspond to the **ImportDeleteFlags** field.

ImportDeleteFlags (1 byte): An 8-bit flag structure that defines the parameters of the import operation.

The following table defines valid flags for the **ImportDeleteFlags** field.

Flag name	Value	Description
Hierarchy	0x01	If this flag is set, folder deletions are being imported. If this flag is not set, message deletions are being imported.
HardDelete	0x02	The server SHOULD**<16>** support this flag. If this flag is set, hard deletions are being imported. If this flag is not set, hard deletions are not being imported.

PropertyValueCount (2 bytes): An unsigned 16-bit integer. This value specifies the number of structures present in the **PropertyValues** field.

PropertyValues (variable): An array of **TaggedPropertyValue** structures ([MS-OXCDATA] section 2.11.4). This value MUST NOT be NULL. The value of this field is used to specify the folders or messages to be deleted.

The following table defines the restrictions that exist on the **PropertyValues** field.

Property	Restrictions	Comments
[**PtypMultipleBinary**] ([MS-OXCDATA] section 2.11.1) 0x00001102	Required Fixed position	An array of serialized **GID** structures ([MS-OXCDATA] section 2.2.1.3) that represents the objects to be deleted.
< other properties >	*Prohibited*	None.

2.2.3.2.4.5.2 RopSynchronizationImportDeletes ROP Response Buffer

The following descriptions define valid fields for the response buffer of the **RopSynchronizationImportDeletes** ROP ([MS-OXCROPS] section 2.2.13.5).

ReturnValue (4 bytes): An unsigned 32-bit integer. This value represents the ROP execution status. For details about common return values for **RopSynchronizationImport*** ROPs that require special processing, see section 3.3.4.3.3.

2.2.3.2.4.6 RopSynchronizationImportReadStateChanges ROP

The **RopSynchronizationImportReadStateChanges** ROP ([MS-OXCROPS] section 2.2.13.3) imports message read state changes into the server replica.

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

For more details about client behaviors related to this ROP, see sections 3.3.4.3.3 and 3.3.5.8.11. For more details about server behaviors related to this ROP, see section 3.2.5.9.4.6.

2.2.3.2.4.6.1 RopSynchronizationImportReadStateChanges ROP Request Buffer

The following descriptions define valid fields for the request buffer of the **RopSynchronizationImportReadStateChanges** ROP ([MS-OXCROPS] section 2.2.13.3).

InputServerObject: The value of this field MUST be the synchronization upload context configured to collect changes to content.

MessageReadStateSize (2 bytes): An unsigned 16-bit integer. This value specifies the size in bytes of the **MessageReadStates** field. The value MUST NOT be set to 0x0000.

MessageReadStates (variable): An array of **MessageReadState** structures ([MS-OXCROPS] section 2.2.13.3.1.1) — one per each message that is changing its read state — that consist of the following:

- **MessageIdSize (2 bytes):** An unsigned 16-bit integer. This value specifies the size of the **MessageId** field. This value MUST NOT be set to 0x0000.
- **MessageId (variable):** An array of bytes. Contains the **XID** structure, as specified in section 2.2.2.2, that represents the **PidTagSourceKey** property (section 2.2.1.2.5) for a message that is changing its read state.

- **MarkAsRead (1 byte):** An 8-bit **PtypBoolean** ([MS-OXCDATA] section 2.11.1). This value specifies whether to mark the message as read (0x01) or unread (0x00).

MID structures ([MS-OXCDATA] section 2.2.1.2) identifying FAI messages in the value of the **MessageReadStates** field are ignored.

2.2.3.2.4.6.2 RopSynchronizationImportReadStateChanges ROP Response Buffer

The following descriptions define valid fields for the response buffer of the **RopSynchronizationImportReadStateChanges** ROP ([MS-OXCROPS] section 2.2.13.3).

ReturnValue (4 bytes): An unsigned 32-bit integer. This value represents the ROP execution status. For details about common return values for the **RopSynchronizationImport*** ROPs that require special processing, see section 3.3.4.3.3.

2.2.3.2.4.7 RopGetLocalReplicaIds ROP

The **RopGetLocalReplicaIds** ROP ([MS-OXCROPS] section 2.2.13.13) allocates a range of internal identifiers for the purpose of assigning them to client-originated objects in a local replica. For more details about client-assigned internal identifiers, see section 3.3.5.2.1.

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

For more details about client behaviors related to this ROP, see sections 3.3.4.3.3 and 3.3.5.8.12. For more details about server behaviors related to this ROP, see section 3.2.5.9.4.7.

2.2.3.2.4.7.1 RopGetLocalReplicaIds ROP Request Buffer

The following descriptions define valid fields for the request buffer of the **RopGetLocalReplicaIds** ROP ([MS-OXCROPS] section 2.2.13.13).

InputServerObject: The value of this field MUST be a **Logon** object.

IdCount (4 bytes): An unsigned 32-bit integer. This value specifies the number of IDs to be allocated.

2.2.3.2.4.7.2 RopGetLocalReplicaIds ROP Response Buffer

The following descriptions define valid fields for the response buffer of the **RopGetLocalReplicaIds** ROP ([MS-OXCROPS] section 2.2.13.13).

ReturnValue (4 bytes): An unsigned 32-bit integer. This value represents the ROP execution status.

REPLGUID (16 bytes): A GUID value that specifies the **REPLGUID** structure shared by all allocated IDs. This value MUST be present if and only if value of the **ReturnValue** field equals **Success** (0x00000000).

GlobalCount (6 bytes): An array of bytes. This array and the value of the **REPLGUID** field are combined to produce the first **GID** structure ([MS-OXCDATA] section 2.2.1.3) in the allocated range of IDs, which is defined as [**GlobalCount**, **GlobalCount** + **IdCount** – 1]. This value MUST be present if and only if the value of the **ReturnValue** field equals **Success** (0x00000000).

2.2.3.2.4.8 RopSetLocalReplicaMidsetDeleted ROP

The **RopSetLocalReplicaMidsetDeleted** ROP ([MS-OXCROPS] section 2.2.13.12) identifies that a set of IDs either belongs to deleted messages in the specified folder or will never be used for any

messages in the specified folder. This ROP is intended for use on IDs that were used on the client and never uploaded to the server, or were never used on the client.

The **RopSetLocalReplicaMidsetDeleted** ROP does not delete IDs from the server; it only reports that they cannot be used within a given folder. To delete IDs and messaging objects from the server, use the **RopSynchronizationImportDeletes** ROP (section 2.2.3.2.4.5).

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

For more details about client behaviors related to this ROP, see sections 3.3.4.3.3 and 3.3.5.8.13. For more details about server behaviors related to this ROP, see section 3.2.5.9.4.8.

2.2.3.2.4.8.1 RopSetLocalReplicaMidsetDeleted ROP Request Buffer

The following descriptions define valid fields for the request buffer of the **RopSetLocalReplicaMidsetDeleted** ROP ([MS-OXCROPS] section 2.2.13.12).

InputServerObject: The value of this field MUST be a Folder object.

DataSize (2 bytes): An unsigned 16-bit integer. This value specifies the size of both the **LongTermIdRangeCount** and **LongTermIdRanges** fields. This value MUST NOT be set to 0x0000.

LongTermIdRangeCount (4 bytes): An unsigned 32-bit integer. This value specifies the number of structures in the **LongTermIdRanges** field. This value MUST NOT be set to 0x00000000.

LongTermIdRanges (variable): An array of **LongTermIdRange** structures. All of the IDs in this field MUST have been obtained previously by using the **RopGetLocalReplicaIds** ROP (section 2.2.3.2.4.7). Each **LongTermIdRange** structure defines a range of IDs, which are reported as unused or deleted on the client. This value consists of the following:

- **MinLongTermId (24 bytes):** A **LongTermID** structure ([MS-OXCDATA] section 2.2.1.3.1) that defines the ID by using the minimum value of the **GLOBCNT** structure, as specified in section 2.2.2.5, that belongs to a range.
- **MaxLongTermId (24 bytes):** A **LongTermID** structure that defines the ID by using the maximum value of the **GLOBCNT** structure that belongs to a range.

The **MinLongTermId** and **MaxLongTermId** fields MUST have the same values for their **REPLGUID** structures.

2.2.3.2.4.8.2 RopSetLocalReplicaMidsetDeleted ROP Response Buffer

The following descriptions define valid fields for the response buffer of the **RopSetLocalReplicaMidsetDeleted** ROP ([MS-OXCROPS] section 2.2.13.12).

ReturnValue (4 bytes): An unsigned 32-bit integer. This value represents the ROP execution status.

2.2.4 FastTransfer Stream

The information set encoded in a FastTransfer stream depends on the type and parameters of the operation that produces it, as specified in section 2.2.4.4. Parsing (syntactic analysis) of the stream can be done without knowing what operation produced it.

At a high level, the FastTransfer stream contains serialized mailbox data and **markers**. Note that markers are not properties and can never have a value, although they are specified in [MS-OXPROPS] and have the same syntax as property tags. The complete list of markers is specified in section 2.2.4.1.4.

Section 2.2.4.1 and section 2.2.4.2 contain an **Augmented Backus-Naur Form (ABNF)** like description of the tokenized FastTransfer stream structure. The description uses the conventions specified in [RFC5234], except for the following:

- For display purposes, indented lines represent a continuation of the lines that precede them.

Despite their name, FastTransfer streams are not represented as Stream objects, and they can only be manipulated by using the **RopFastTransferSourceGetBuffer** ROP (section 2.2.3.1.1.5) for download operations and **RopFastTransferDestinationPutBuffer** ROP (section 2.2.3.1.2.2) for upload operations. For more details about how FastTransfer streams are produced and processed by ROPs, see section 2.2.4.4.

2.2.4.1 Lexical structure

The lexical structure of the FastTransfer stream is essential to let its producers and consumers agree on rules that govern splitting of the stream into sequential buffers retrieved by using the **RopFastTransferSourceGetBuffer** ROP (section 2.2.3.1.1.5) or supplied through the **RopFastTransferDestinationPutBuffer** ROP (section 2.2.3.1.2.2). It is also beneficial for an explanation of the protocol, as it separates matters of data serialization and deserialization (lexical analysis) from data and data organization (syntactical analysis), and from its mapping to mailbox concepts (semantics).

The lexical structure of a FastTransfer stream is as follows:

```
stream            = 1*element
element           = marker / propValue
marker            = PtypInteger32 <from the table in 2.2.4.1.4>
propValue         = fixedPropType propInfo fixedSizeValue
propValue         =/ varPropType propInfo length varSizeValue
propValue         =/ mvPropType
                       propInfo
                       length
                       *( fixedSizeValue / length varSizeValue )
propInfo          = taggedPropId / ( namedPropId namedPropInfo )
fixedSizeValue    = PtypInteger16 / PtypInteger32 / PtypFloating32
                    / PtypFloating64 / PtypCurrency / PtypFloatingTime
                    / PtypBoolean / PtypInteger64 / PtypTime
                    / PtypGuid
varSizeValue      = PtypString / PtypString8 / PtypServerId
                    / PtypBinary / PtypObject


namedPropInfo     = propertySet
                     ((%x00 dispid)
                     / (%x01 name))
propertySet        = PtypGuid
dispid             = PtypInteger32
name               = PtypString
namedPropId       = propertyId
                     <Greater or equal to 0x8000>
propertyId        = PtypInteger16
taggedPropId       = propertyId
                     <less than 0x8000>
length            = PtypInteger32 <MUST be greater than 0>
propType          = fixedPropType / varPropType / mvPropType
fixedPropType     = PtypInteger16
varPropType       = PtypInteger16
mvPropType        = PtypInteger16
```

For more details about the **fixedPropType**, **varPropType**, and **mvPropType** **property types**, see section 2.2.4.1.1.

The lexical structure of the FastTransfer adheres to the following guidelines:

- **Camel-cased** names are nonterminal syntactic elements, as specified in [RFC5234] section 2.3.
- **Pascal-cased** names with a **Ptyp** prefix are any value of that type serialized as specified in section 2.2.4.1.3.

A FastTransfer stream can be larger than a single buffer. The server MUST split the stream when it cannot fit into a single buffer. If a split is required, the stream MUST be split either between two atoms or at any point inside a **varSizeValue** lexeme. A stream MUST NOT be split within a single atom. The lexical structure of an atom is as follows:

```
atom                = marker
                    / propDef
                    / fixedSizeValue
                    / length
propDef             = ( propType propInfo )
```

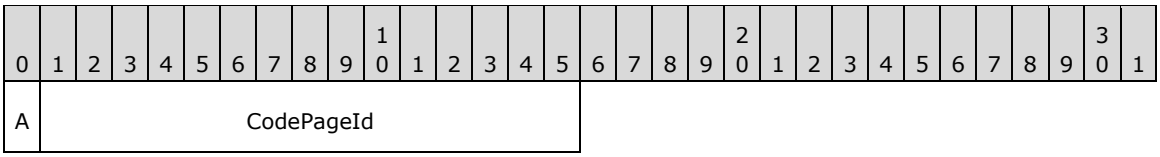
2.2.4.1.1 fixedPropType, varPropType, mvPropType Property Types

Property types supported in FastTransfer streams are a subset of those defined in [MS-OXCDATA] section 2.11.1.

Property type	Description
fixedPropType	Property type value of any type that has a fixed length, as specified in [MS-OXCDATA] section 2.11.1.
varPropType	Property type value of **PtypString**, **PtypString8**, **PtypBinary**, **PtypServerId**, **PtypObject** ([MS-OXCDATA] section 2.11.1), or a code page string property type, as specified in section 2.2.4.1.1.1.
mvPropType	Property type value of any multi-valued property type (starts with **PtypMultiple** ([MS-OXCDATA] section 2.11.1)), whose base type is either a valid **fixedPropType** or a valid **varPropType**.

2.2.4.1.1.1 Code Page Property Types

Code page property types are a 2-byte value used to transmit string properties using the code page format of the string as stored on the server, in server-to-client-to-server scenarios. For example, a code page property type of 0x84B0 specifies the Unicode (1200) code page and a code page property type of 0x84E2 specifies the Western European (1250) code page.



A (1 bit): 1-bit flag (mask 0x8000). This bit MUST be set to 1 to indicate the property is an internal code page string.

CodePageId (15 bit): The decimal value of the code page identifier for the code page used to encode the string property.

2.2.4.1.2 propValue Lexical Element

The **propValue** element represents the identification and the value of a property or a meta-property.

The **fixedSizeValue** or **varSizeValue** lexemes contained in a **propValue** element represent the value of the property and MUST be serializations of a **base property type** for a property type specified with contained **fixedPropType**, **varPropType**, or **mvPropType** property type values.

2.2.4.1.3 Serialization of Simple Types

Serialization of simple types in FastTransfer streams is identical to serialization of property values as specified [MS-OXCDATA], with the following exceptions:

Property type name	Difference in serialization
PtypBoolean ([MS-OXCDATA] section 2.11.1)	2-bytes in FastTransfer streams, instead of 1-byte as specified in [MS-OXCDATA]. Using **little-endian** byte ordering, "01 00" for **TRUE** and "00 00" for **FALSE**.
PtypString **PtypString8** ([MS-OXCDATA] section 2.11.1)	Serialization MUST be performed, as specified in [MS-OXCDATA]. The server SHOULD<17> output string values with the terminating nulls. FastTransfer stream readers MUST check that the last 1 (for **PtypString8**) or 2 (for **PtypString**) bytes of a stream are indeed zeros before truncating them.

Note that little-endian byte ordering MUST be used. The data type of simple type elements determine how bytes are serialized on the wire. For example, Int16 value 0x1234 is encoded as "34 12" on the wire.

2.2.4.1.4 Markers

The following table shows the complete list of markers used in FastTransfer streams. The **PidTag** prefix is omitted in the ABNF specified in section 2.2.4.2 to emphasize their difference from properties.

Start/stand-alone marker name and its numeric value		Corresponding end marker , if applicable, and its numeric value	
Folders			
StartTopFld	0x40090003	**EndFolder**	0x400B0003
StartSubFld	0x400A0003		
Messages and their parts			
StartMessage	0x400C0003	**EndMessage**	0x400D0003
StartFAIMsg	0x40100003		
StartEmbed	0x40010003	**EndEmbed**	0x40020003
StartRecip	0x40030003	**EndToRecip**	0x40040003
NewAttach	0x40000003	**EndAttach**	0x400E0003
Synchronization download			
IncrSyncChg	0x40120003	None.	
IncrSyncChgPartial	0x407D0003	None.	
IncrSyncDel	0x40130003	None.	
IncrSyncEnd	0x40140003	None.	
IncrSyncRead	0x402F0003	None.	

Start/stand-alone marker name and its numeric value		Corresponding end marker , if applicable, and its numeric value	
IncrSyncStateBegin	0x403A0003	**IncrSyncStateEnd**	0x403B0003
IncrSyncProgressMode	0x4074000B	None.	
IncrSyncProgressPerMsg	0x4075000B	None.	
IncrSyncMessage	0x40150003	None.	
IncrSyncGroupInfo	0x407B0102	None.	
Special			
FXErrorInfo	0x40180003	None.	

The **StartTopFld** marker signifies the start of data that describes a folder.

The **StartSubFld** marker signifies the start of serialized data that describes a mailbox subfolder.

The **EndFolder** marker signifies the end of serialized data that describes a mailbox folder or subfolder.

The **StartMessage** marker signifies the start of serialized data that describes an e-mail message.

The **StartFAIMsg** marker signifies the start of serialized data that describes an FAI message.

The **EndMessage** marker signifies the end of serialized data that describes an e-mail message.

The **StartEmbed** marker signifies the start of an embedded e-mail message.

The **EndEmbed** marker signifies the end of an embedded e-mail message.

The **StartRecip** marker signifies the start of recipient data.

The **EndToRecip** marker signifies the end of recipient data.

The **NewAttach** marker signifies the start of an attachment.

The **EndAttach** marker signifies the end of an attachment.

The **IncrSyncChg** marker signifies the start of ICS information pertaining to the message.

The **IncrSyncChgPartial** marker signifies the start of data that describes the property group mapping for properties that have changed in a partial message.

The **IncrSyncDel** marker signifies the start of deleted message data in the stream.

The **IncrSyncEnd** marker signifies the end of serialized ICS data.

The **IncrSyncRead** marker signifies the start of serialized data that describes which messages are to be marked as read or unread.

The **IncrSyncStateBegin** marker signifies the start of data that describes the synchronization state after ICS finishes.

The **IncrSyncStateEnd** marker signifies the end of serialized data that describes the synchronization state after ICS finishes.

The **IncrSyncProgressMode** marker signifies the start of serialized data that describes the size of all the ICS data to be transmitted.

The **IncrSyncProgressPerMsg** marker signifies the start of the serialized data that describes the size of the next message in the stream.

The **IncrSyncMessage** marker signifies the start of e-mail data for ICS.

The **IncrSyncGroupInfo** marker signifies the start of data that describes property group mapping information.

The **FXErrorInfo** marker signifies the start of error data.

2.2.4.1.5 Meta-Properties

Meta-properties contain information about how to process data, instead of containing data to be processed. Use of meta-properties specified in this section is restricted to specific occasions in FastTransfer streams; therefore, values for these meta-properties are serialized according to FastTransfer stream rules, as specified in section 2.2.4.1.3.

2.2.4.1.5.1 MetaTagFXDelProp Meta-Property

Property ID: 0x4016

Data type: **PtypInteger32**, 0x0003 ([MS-OXCDATA] section 2.11.1)

The **MetaTagFXDelProp** meta-property represents a directive to a client to delete specific subobjects of the object in context. The type of subobjects to delete is determined by the value of the meta-property, which can be any of the property tags specified in section 2.2.1.7.

2.2.4.1.5.2 MetaTagEcWarning Meta-Property

Property ID: 0x400F

Data type: **PtypInteger32**, 0x0003 ([MS-OXCDATA] section 2.11.1)

The **MetaTagEcWarning** meta-property contains a warning that occurred when producing output for an element in context.

The following error code requires special processing when passed as a value of the **MetaTagEcWarning** meta-property:

Error code name	Description
PartiallyComplete	The client SHOULD verify that properties and subobjects of the object represented by an element in context were output completely.

The complete list of error codes is specified in [MS-OXCDATA] section 2.4.

2.2.4.1.5.3 MetaTagNewFXFolder Meta-Property

Property ID: 0x4011

Data type: **PtypBinary**, 0x0102 ([MS-OXCDATA] section 2.11.1)

The **MetaTagNewFXFolder** meta-property provides information about alternative replicas (1) for a public folder in context. This meta-property represents a serialized **FolderReplicaInfo** structure, as specified in section 2.2.2.9.

2.2.4.1.5.4 MetaTagIncrSyncGroupId Meta-Property

Property ID: 0x407C

Data type: **PtypInteger32**, 0x0003 ([MS-OXCDATA] section 2.11.1)

The **MetaTagIncrSyncGroupId** meta-property specifies an identifier of a property group mapping. This value directs the client to use the specified property group mapping where applicable, until the value is reset with another instance of the **MetaTagIncrSyncGroupId** meta-property.

For more details about property groups, see section 3.2.5.7.

2.2.4.1.5.5 MetaTagIncrementalSyncMessagePartial Meta-Property

Property ID: 0x407A

Data type: **PtypInteger32**, 0x0003 ([MS-OXCDATA] section 2.11.1)

The **MetaTagIncrementalSyncMessagePartial** meta-property specifies an index of a property group within a property group mapping currently in context, to be used for partial item downloads. This meta-property instructs the client to read all forthcoming property values as a part of the specified group, where applicable, until reset with another instance of the **MetaTagIncrementalSyncMessagePartial** meta-property.

For more details about property groups, see section 3.2.5.7.

2.2.4.1.5.6 MetaTagDnPrefix Meta-Property

Property ID: 0x4008

Data type: **PtypString8**, 0x001E ([MS-OXCDATA] section 2.11.1)

The **MetaTagDnPrefix** meta-property MUST be ignored when received.

2.2.4.2 Syntactical Structure

The syntactical structure of the FastTransfer adheres to the following guidelines:

- Camel-cased names are nonterminal syntactic elements, as specified in [RFC5234] section 2.3.
- Pascal-cased names without a **PidTag** prefix are markers. Markers are specified in section 2.2.4.1.4.
- Pascal-cased names with a **PidTag** prefix are properties and are defined in [MS-OXPROPS].
- Pascal-cased names with a **MetaTag** prefix are **meta-properties**. Meta-properties are specified in section 2.2.4.1.5.

Note that markers never have a value, and meta-properties, just as regular properties, always have a value when serialized into a FastTransfer stream. Therefore, wherever a marker exists, it is serialized as 4 bytes. Meta-properties, on the other hand, are serialized the same as **propValue** elements.

The syntactical structure of a FastTransfer stream is as follows:

```
root                = contentsSync
                    / hierarchySync
                    / state
                    / folderContent
                    / messageContent
                    / attachmentContent
                    / messageList
```

```
                    / topFolder

propValue           = <see lexical structure in 2.2.4.1>
errorInfo           = FXErrorInfo propList
propList            = *propValue

subFolder           = StartSubFld folderContent EndFolder
subFolderNoDelProps = StartSubFld folderContentNoDelProps EndFolder
topFolder           = StartTopFld folderContentNoDelProps EndFolder
folderContent       = propList
                    ( MetaTagNewFXFolder / folderMessages )
                    [ MetaTagFXDelProp *subFolder ]
folderContentNoDelProps = propList
                    ( MetaTagNewFXFolder / folderMessagesNoDelProps )
                    [ *subFolderNoDelProps ]
folderMessages      = *2( MetaTagFXDelProp messageList )
folderMessagesNoDelProps = *2( messageList )
message             = ( StartMessage / StartFAIMsg )
                    messageContent
                    EndMessage
messageChildren     = [ MetaTagFXDelProp ] [ *recipient ]
                    [ MetaTagFXDelProp ] [ *attachment ]
messageContent      = propList messageChildren
messageList         = *( [MetaTagEcWarning] message )
recipient           = StartRecip propList EndToRecip

attachment          = NewAttach PidTagAttachNumber attachmentContent EndAttach
attachmentContent   = propList [embeddedMessage]
embeddedMessage     = StartEmbed messageContent EndEmbed

contentsSync        = [progressTotal]
                    *( [progressPerMessage] messageChange )
                    [deletions]
                    [readStateChanges]
                    state
                    IncrSyncEnd
hierarchySync       = *folderChange
                    [deletions]
                    state
                    IncrSyncEnd
deletions           = IncrSyncDel propList
folderChange        = IncrSyncChg propList
groupInfo           = IncrSyncGroupInfo propList
messageChange       = messageChangeFull / messageChangePartial
messageChangeFull   = IncrSyncChg messageChangeHeader
                    IncrSyncMessage propList
                    messageChildren
messageChangeHeader = propList
messageChangePartial = groupInfo MetaTagIncrSyncGroupId
                    IncrSyncChgPartial messageChangeHeader
                    *( MetaTagIncrementalSyncMessagePartial propList )
                    messageChildren
progressPerMessage  = IncrSyncProgressPerMsg propList
progressTotal       = IncrSyncProgressMode propList
readStateChanges    = IncrSyncRead propList
state               = IncrSyncStateBegin propList IncrSyncStateEnd
```

2.2.4.3 Semantics of Elements

2.2.4.3.1 attachmentContent Element

The **attachmentContent** element contains the properties and the Embedded Message object of an Attachment object, if present.

Property filters, as specified in section 3.2.5.10, can affect the Attachment object properties contained in the **propList** element, as specified in section 2.2.4.3.20.

2.2.4.3.2 contentsSync Element

The **contentsSync** element contains the result of the content synchronization download operation.

For details about how servers determine the set of differences to be downloaded to clients, see section 3.2.5.3.

2.2.4.3.3 deletions Element

The **deletions** element contains information about IDs of messaging objects that have been deleted, expired, or moved out of the synchronization scope since the last synchronization, as specified in the initial ICS state. For details about how servers determine the set of IDs to be reported by using this element, which is a subset of the IDs in the **deleted item list**, see section 3.2.5.3.

Deletions SHOULD NOT be present if the **NoDeletions** flag of the **SynchronizationFlags** field, as specified in section 2.2.3.2.1.1.1, was set when the synchronization download operation was configured.

The following restrictions exist on the contained **propList** element, as specified in section 2.2.4.3.20:

- MUST contain at least one property.
- MUST adhere to the following restrictions:

Property name	Restrictions	Comments
MetaTagIdsetDeleted (section 2.2.1.3.1)	No restrictions	None.
MetaTagIdsetNoLongerInScope (section 2.2.1.3.2)	Conditional	MUST be present if the **Contents** value of the **SynchronizationType** field is set and there are Message objects that moved out of the synchronization scope since the last synchronization. MUST NOT be present if the **Hierarchy** value of the **SynchronizationType** field is set, as specified in section 2.2.3.2.1.1.1. MUST NOT be present if the **IgnoreNoLongerInScope** flag of the **SynchronizationFlags** field is set.
MetaTagIdsetExpired (section 2.2.1.3.3)	Conditional	MUST be present if the **Contents** value of the **SynchronizationType** field is set and the Message objects in a public folder expired since the last synchronization. MUST NOT be present if the **Hierarchy** value of the **SynchronizationType** field is set.
< other properties >	*Prohibited*	None.

2.2.4.3.4 errorInfo Element

The **errorInfo** element provides out-of-band error reporting and recovery. It is used to provide support for partial completion of the operations by scoping the failures down to the failing object, rather than the entire operation.

The **errorInfo** element is inserted into the stream whenever the server internally encounters an error retrieving information or building the necessary information to serialize the messaging object. The **errorInfo** element can be inserted wherever a lexical structure, specified in section 2.2.4.1, allows a marker or a **propValue** element.

This element SHOULD be used if and only if the **RecoverMode** flag of the **SendOptions** field is set. Note that by the time a server encounters an error that requires failing the download of a messaging object in context, it might have already output some part of the data pertaining to that object in the previous buffer.

Clients MUST support parsing of this element if the client set the **RecoverMode** flag in the **SendOptions** field.

Whenever a server or a client produces or parses this element, it MUST unwind its producing or parsing stack up to, but not including, the closest element that supports recovery. The current version of the protocol defines two such elements: **contentsSync**, as specified in section 2.2.4.3.2, and **messageList**, as specified in section 2.2.4.3.17. Upon receiving this element, clients can perform additional steps to remove a faulty object from future synchronizations, as described in section 3.3.5.10.

The following table lists the restrictions that exist on the contained **propList** element, as specified in section 2.2.4.3.20.

Property type name	Restrictions	Comments
[**PtypBinary**] ([MS-OXCDATA] section 2.11.1) 0x00000102	Required Fixed position	Serialized **ExtendedErrorInfo** structure, as specified in section 2.2.2.10.
< other properties >	*Prohibited*	None.

2.2.4.3.5 folderChange Element

The **folderChange** element contains a new or changed folder in the hierarchy synchronization.

The **propList** element, as specified in section 2.2.4.3.20, that is contained in the **folderChange** element includes the properties of the Folder object, possibly affected by property filters (as specified in section 3.2.5.10) and combined with additional mandatory properties that are required for object identification and conflict detection.

The following table lists the restrictions that exist on the contained **propList** element, as specified in section 2.2.4.3.20.

Property name	Restrictions	Comments
PidTagParentSourceKey (section 2.2.1.2.6)	Required	None.
PidTagSourceKey (section 2.2.1.2.5)	Required	None.
PidTagLastModificationTime ([MS-OXPROPS] section 2.755)	Required	None.
PidTagChangeKey (section 2.2.1.2.7)	Required	None.
PidTagPredecessorChangeList (section 2.2.1.2.8)	Required	None.
PidTagDisplayName ([MS-OXCFOLD] section 2.2.2.2.2.5)	Required	None.

Property name	Restrictions	Comments
PidTagFolderId (section 2.2.1.2.2)	Conditional	MUST be present if and only if the **Eid** flag of the **SynchronizationExtraFlags** field is set.
PidTagParentFolderId (section 2.2.1.2.4)	Conditional	MUST be present if the **NoForeignIdentifiers** flag of the **SynchronizationFlags** field is set. SHOULD<18> be present if the **Eid** flag of the **SynchronizationExtraFlags** field is set.
< other properties >	No restrictions	None.

2.2.4.3.6 folderContent Element

The **folderContent** element contains the content of a folder: its properties, messages, and subfolders.

The **propList** element, as specified in section 2.2.4.3.20, contains the properties of the Folder object, which are possibly affected by property filters, as specified in section 3.2.5.10.

The following table lists the restrictions that exist on the contained **propList** element.

Property name	Restrictions	Comments
PidTagFolderId (section 2.2.1.2.2)	Conditional Fixed position	MUST be present if and only if the first marker in the **folder** element is not the **StartTopFld** marker, as specified in section 2.2.4.1.4.
PidTagDisplayName ([MS-OXCFOLD] section 2.2.2.2.2.5)	Conditional Fixed position	MUST be present if and only if the first marker in the **folder** element is not the **StartTopFld** marker
PidTagComment ([MS-OXCFOLD] section 2.2.2.2.2.2)	Conditional Fixed position	MUST be present if and only if the first marker in the **folder** element is not the **StartTopFld** marker
MetaTagEcWarning (section 2.2.4.1.5.2)	Conditional Fixed position	MAY<19> be output by the server if the client set the **Move** flag of the **CopyFlags** field and the user does not have permissions to delete the source folder.
< other properties >	No restrictions	None.

For more details about the impact of property and subobject filters that are specified when configuring an operation on the content of this element, see section 3.2.5.10.

The **MetaTagNewFXFolder** meta-property (section 2.2.4.1.5.3) MUST be output instead of message elements when outputting a public folder whose contents do not exist on the server because the content is ghosted. If there is a valid replica (1) of the public folder on the server and the folder content has not replicated to the server yet, the folder content is not included in the FastTransfer stream as part of the **folderContent** element. The server SHOULD NOT include any data following the **MetaTagNewFXFolder** meta-property in the buffer returned by the **RopFastTransferSourceGetBuffer** ROP (section 2.2.3.1.1.5), although additional data can be included in the FastTransfer stream. Any data included after this property in the buffer returned by the **RopFastTransferSourceGetBuffer** ROP is ignored by the client, which results in a parsing failure when the client attempts to parse the next buffer.

Under conditions specified in section 3.2.5.10, the **PidTagContainerHierarchy** property ([MS-OXPROPS] section 2.636) included in a **subFolder** element MUST be preceded by a **MetaTagFXDelProp** meta-property (section 2.2.4.1.5.1).

2.2.4.3.7 folderMessages Element

The **folderMessages** element contains the messages contained in a folder.

Under conditions specified in section 3.2.5.10, when included in the **folderMessages** element, the **PidTagFolderAssociatedContents** ([MS-OXPROPS] section 2.690) and **PidTagContainerContents** ([MS-OXPROPS] section 2.634) properties MUST be preceded by a **MetaTagFXDelProp** meta-property (section 2.2.4.1.5.1).

2.2.4.3.8 groupInfo Element

The **groupInfo** element provides a definition for the property group mapping, as specified in section 3.2.5.7. Property group mappings, after they are defined by using the **groupInfo** element, can be referenced with the **MetaTagIncrSyncGroupId** meta-property further in the stream by its group ID.

The following table lists the restrictions that exist on the contained **propList** element, as specified in section 2.2.4.3.20.

Property type name	Restrictions	Comments
[**PtypBinary**] ([MS-OXCDATA] section 2.11.1) 0x00000102	Required Fixed position	Serialized **PropertyGroupInfo** structure, as specified in section 2.2.2.8.
< other properties >	*Prohibited*	None.

2.2.4.3.9 hierarchySync Element

The **hierarchySync** element contains the result of the hierarchy synchronization download operation.

There MUST be exactly one **folderChange** element for each descendant folder of the root of the synchronization operation (that is the folder that was passed to the **RopSynchronizationConfigure** ROP, as specified in section 2.2.3.2.1.1) that is new or has been changed since the last synchronization. The **folderChange** elements for the parent folders MUST be output before any of their child folders.

The parent-child relationship is determined by comparing the **PidTagSourceKey** property (section 2.2.1.2.5) of a prospective parent folder and a **PidTagParentSourceKey** property (section 2.2.1.2.6) of a prospective child folder. The **folderChange** elements that have a **PidTagParentSourceKey** property with a zero-length value are children of the root of the synchronization operation.

For details about how servers determine the set of differences to be downloaded to clients, see section 3.2.5.3.

2.2.4.3.10 message Element

The **message** element represents a Message object.

The type of the starting marker to use depends on whether the message is a normal message or an FAI message. Normal messages use the **StartMessage** marker; FAI messages use the **StartFAIMsg** marker. For more details about markers, see section 2.2.4.1.4.

2.2.4.3.11 messageChange Element

The **messageChange** element represents a change to a Message object.

A server MUST use the **messageChangeFull** element, instead of the **messageChangePartial** element, if any of the following are true:

- The **PartialItem** flag of the **SendOptions** field was not set, as specified in section 2.2.3.2.1.1.
- The **Message ID** structure ([MS-OXCDATA] section 2.2.1.2) of the message to be output is not in the **MetaTagIdsetGiven** property (section 2.2.1.1.1) from the initial ICS state.
- The message is an FAI message.
- The message is a conflicting version contained in a conflict resolve message. For more details, see section 3.1.5.6.2.1.

Otherwise, the server determines the most efficient way to communicate the message change on a case-by-case basis.

2.2.4.3.12 messageChildren Element

The **messageChildren** element represents child objects of the Message objects: **Recipient objects** and Attachment objects.

For details about the impact of property and subobject filters that are specified when configuring an operation on the content of this element, see section 3.2.5.10.

Under the conditions specified in section 3.2.5.10, the **PidTagMessageRecipients** ([MS-OXPROPS] section 2.786) property included in a recipient element and the **PidTagMessageAttachments** ([MS-OXPROPS] section 2.776) property included in an **attachment** element MUST be preceded by a **MetaTagFXDelProp** meta-property (section 2.2.4.1.5.1)and .

2.2.4.3.13 messageChangeFull Element

The **messageChangeFull** element contains the complete content of a new or changed message: the message properties, the recipients, and the attachments.

Property filters, as specified in section 3.2.5.10, can affect the Message object properties in the contained **propList** element, as specified in section 2.2.4.3.20.

2.2.4.3.14 messageChangeHeader Element

The **messageChangeHeader** element contains a fixed set of information about the message change that follows this element in the FastTransfer stream. The information in the header is sufficient for message identification and conflict detection.

The following table lists the restrictions that exist on the contained **propList** element, as specified in section 2.2.4.3.20.

Property name	Restrictions	Comments
PidTagSourceKey (section 2.2.1.2.5)	Required Fixed position	None.
PidTagLastModificationTime ([MS-OXPROPS] section 2.755)	Required Fixed position	None.

Property name	Restrictions	Comments
PidTagChangeKey (section 2.2.1.2.7)	Required Fixed position	None.
PidTagPredecessorChangeList (section 2.2.1.2.8)	Required Fixed position	None.
PidTagAssociated (section 2.2.1.5)	Required Fixed position	None.
PidTagMid (section 2.2.1.2.1)	Conditional	MUST be present if and only if the **Eid** flag of the **SynchronizationExtraFlags** field is set, as specified in section 2.2.3.2.1.1.1.
PidTagMessageSize (section 2.2.1.6)	Conditional	MUST be present if and only if the **MessageSize** flag of the **SynchronizationExtraFlags** field is set.
PidTagChangeNumber (section 2.2.1.2.3)	Conditional	MUST be present if and only if the **CN** flag of the **SynchronizationExtraFlags** field is set.
< other properties >	*Prohibited*	None.

2.2.4.3.15 messageChangePartial Element

The **messageChangePartial** element represents the difference in message content since the last download, as identified by the initial ICS state. Servers SHOULD<20> support partial item downloads. Changes to a message are output based on the granularity of the property group, as specified in section 3.2.5.7. The value of the last encountered **MetaTagIncrSyncGroupId** meta-property (section 2.2.4.1.5.4) in the stream determines which property group mapping MUST be used to make partial updates to the messaging object. The **MetaTagIncrSyncGroupId** meta-property identifies the **groupInfo** element in the **messageChangePartial** element that organizes properties into property groups for the messaging object in context.

The **groupInfo** element, as specified in section 2.2.4.3.8, when included in the **messageChangePartial** element is not required to be the property group definition of the message that follows or the definition indicated by the **MetaTagIncrSyncGroupId** meta-property.

Clients MUST treat every contained **propList** element, as specified in section 2.2.4.3.20, as the complete content of a property group denoted by the **MetaTagIncrementalSyncMessagePartial** meta-property (section 2.2.4.1.5.5) that preceded it. That is, all properties missing from the **propList** element, but defined for this group in the corresponding property group mapping, MUST be deleted from the Message object. For example, if the value for the preceding **MetaTagIncrementalSyncMessagePartial** meta-property is 5, the **propList** element for the **messageChangePartial** element contains all the values of for the properties in group 5. Any properties in group 5 that are not in the **propList** of the **messageChangePartial** element are deleted from the Message object.

The following table lists the restrictions that exist on the contained **propList** elements.

Property type name	Restrictions	Comments
[**PtypInteger32**] ([MS-OXCDATA] section 2.11.1) 0x00000003	Conditional	MUST be present if and only if a property group is empty, but was still marked as changed since the last download.

Property type name	Restrictions	Comments
		MUST be 0. MUST be ignored by clients.
[**PtypInteger32**] 0xFFFFFFFF	Conditional	MUST be present if the properties in the properties that follow don't exist in any specific property group in the property group mapping.
< other properties >	No restrictions	None.

2.2.4.3.16 messageContent Element

The **messageContent** element represents the content of a message: its properties, the recipients, and the attachments.

Property filters, as specified in section 3.2.5.10, can affect the Message object properties in the contained **propList** element, as specified in section 2.2.4.3.20.

Property name	Restrictions	Comments
PidTagMid (section 2.2.1.2.1)	Conditional Fixed position	MUST be present in FastTransfer streams created by the **RopFastTransferSourceCopyMessages** (section 2.2.3.1.1.3) and **RopFastTransferSourceCopyFolder** (section 2.2.3.1.1.4) ROPs. Clients MUST ignore the value of this property for Embedded Message objects.
< other properties >	No restrictions	None.

2.2.4.3.17 messageList Element

The **messageList** element contains a list of messages, which is determined by the scope of the operation.

For each message in the **messageList**, the server SHOULD output **MetaTagEcWarning** meta-property (section 2.2.4.1.5.2) if a client does not have the permissions necessary to access it, as specified in section 3.2.5.8.1. The warning is necessary to make it possible for a client to tell this case from a missing message.

2.2.4.3.18 progressPerMessage Element

The **progressPerMessage** element contains data that describes the approximate size of message change data that follows.

MUST be present if and only if the **progressTotal** element, as specified in section 2.2.4.3.18, was output within the same ancestor **contentsSync** element, as specified in section 2.2.4.3.2.

MUST NOT be present if the **Progress** flag of the **SynchronizationFlags** field was not set when configuring the synchronization download operation.

The following table lists the restrictions that exist on the contained **propList** element, as specified in section 2.2.4.3.20.

Property type name	Restrictions	Comments
[**PtypInteger32**] ([MS-OXCDATA] section 2.11.1) 0x00000003	Required Fixed position	Size of the message to be follow. Servers can supply the same value as the **PidTagMessageSize** property (section 2.2.1.6) in the **messageChangeHeader** element, or use a different approximation.
[**PtypBoolean**] ([MS-OXCDATA] section 2.11.1) 0x0000000B	Required Fixed position	**TRUE** (0x0001 or any non-zero value) if the Message object that follows is FAI; otherwise, **FALSE** (0x0000). For more details about the serialization of **PtypBoolean** values in FastTransfer streams, see section 2.2.4.1.3.
< other properties >	*Prohibited*	None.

2.2.4.3.19 progressTotal Element

The **progressTotal** element contains data that describes the approximate size of all the **messageChange** elements, as specified in section 2.2.4.3.11, that will follow in this stream. This element can be used by clients to display progress information. Servers can use a sum of message sizes (**PidTagMessageSize** property (section 2.2.1.6)) for all message changes downloaded in the current operation, or servers can use a different approximation.

Note that this method of reporting progress is provided in addition to what is available in the **RopFastTransferSourceGetBuffer** ROP response, as specified in section 2.2.3.1.1.5. This method of reporting is supposed to reflect the amount of work more precisely, as it is based on message sizes, rather than object count.

This element MUST be present if the **Progress** flag of the **SynchronizationFlags** field, as specified in section 2.2.3.2.1.1.1, was set when configuring the synchronization download operation and a server supports progress reporting.

This element MUST NOT be present if the **Progress** flag of the **SynchronizationFlags** field was not set when configuring the synchronization download operation.

The following table lists the restrictions that exist on the contained **propList** element, as specified in section 2.2.4.3.20.

Property type name	Restrictions	Comments
[**PtypBinary**] ([MS-OXCDATA] section 2.11.1) 0x00000102	Required Fixed position	Serialized **ProgressInformation** structure. For more details, see section 2.2.2.7.
< other properties >	*Prohibited*	None.

2.2.4.3.20 propList Element

The **propList** elements SHOULD NOT<21> contain **propValue** elements for **meta-properties**. All instances in which meta-properties, as specified in section 2.2.4.1.5, can be encountered in a document are mentioned explicitly in the syntax ABNF specified in section 2.2.4.2.

Syntactic elements that contain a **propList** element can express restrictions on a set of properties and/or the position of properties within a list by using property list restriction table syntax, as specified in section 2.2.

Properties that contain an error (have the **PtypErrorCode** type, as specified in [MS-OXCDATA] section 2.11.1) instead of an actual value MUST be omitted from the **propList** element.

2.2.4.3.21 propValue Element

The **propValue** element represents identification information and the value of the property.

Note that the protocol imposes no limit on the size of data that can be encoded using this element, unlike the response buffers of the **RopQueryRows** ROP ([MS-OXCROPS] section 2.2.5.4) and the **RopGetPropertiesSpecific** ROP ([MS-OXCROPS] section 2.2.8.3). Clients and servers MUST fail the operation if the size of data being sent or received is larger than the maximum size imposed by an implementation, rather than truncating the data. For details about the maximum size of a FastTransfer stream for upload, see section 2.2.3.1.2.2.1. For details about the maximum size of a FastTransfer stream for download, see section 3.2.5.8.1.5.

2.2.4.3.22 readStateChanges Element

The **readStateChanges** element contains information about **Message ID** structures ([MS-OXCDATA] section 2.2.1.2) of Message objects that had their read state changed since the last synchronization, as specified by the initial ICS state. For details about how servers determine the set of IDs to be reported by using this element, see section 3.2.5.3.

This element MUST be present if there are changes to the read state of messages This element SHOULD NOT be present if the **ReadState** flag of the **SynchronizationFlags** field was not set when configuring the synchronization download operation.

The following restrictions exist on the contained **propList** element, as specified in section 2.2.4.3.20:

- MUST contain at least one property.
- MUST adhere to the following restrictions:

Property name	Restrictions	Comments
MetaTagIdsetRead (section 2.2.1.3.4)	No restrictions	None.
MetaTagIdsetUnread (section 2.2.1.3.5)	No restrictions	None.
< other properties >	*Prohibited*	None.

2.2.4.3.23 recipient Element

The **recipient** element represents a Recipient object, which is a subobject of the Message object.

The **propList** child element, as specified in section 2.2.4.3.20, contains the properties of the Recipient object.

The following table lists the restrictions that exist on the contained **propList** element.

Property name	Restrictions	Comments
PidTagRowid ([MS-OXPROPS] section 2.930)	Required Fixed position	None.
< other properties >	No restrictions	None.

2.2.4.3.24 root Element

The **root** element contains the root element of FastTransfer streams.

Producers of the FastTransfer stream MUST choose a contained element to generate depending on the Bulk Data Transfer operation in effect. For more details, see the mapping specified in section 2.2.4.4 and section 2.2.3.1.2.1.1.

2.2.4.3.25 state Element

The **state** element contains the final ICS state of the synchronization download operation. For details about how servers construct the final ICS state, see sections 3.1.5.3 and 3.2.5.3.

The following table lists the restrictions that exist on the contained **propList** element, as specified in section 2.2.4.3.20.

Property name	Restrictions	Comments
MetaTagIdsetGiven (section 2.2.1.1.1)	No restrictions	None.
MetaTagCnsetSeen (section 2.2.1.1.2)	No restrictions	None.
MetaTagCnsetSeenFAI (section 2.2.1.1.3)	Conditional	MUST NOT be present if the **SynchronizationType** field is set to **Hierarchy** (0x02), as specified in section 2.2.3.2.1.1.1.
MetaTagCnsetRead (section 2.2.1.1.4)	Conditional	MUST NOT be present if the **SynchronizationType** field is set to **Hierarchy** (0x02).
< other properties >	*Prohibited*	None.

2.2.4.4 FastTransfer Streams in ROPs

The following table describes how possible root elements in the FastTransfer stream correspond to Bulk Data Transfer operations defined in section 2.2.3. Every download operation has to be configured prior to being able to produce a FastTransfer stream. Configuration starts by sending one of the ROPs in the following table and then performing the additional ROP specific configuration steps, as specified in sections 3.3.4.1 and 3.3.4.3.2.

ROP that initiates an operation	Root element in the produced FastTransfer stream	ROP request buffer field conditions
RopSynchronizationConfigure	**contentsSync**	The **SynchronizationType** field is set to **Contents** (0x01).
	hierarchySync	The **SynchronizationType** field is set to **Hierarchy** (0x02).
RopSynchronizationGetTransferState	**state**	Always.
RopFastTranserSourceCopyTo **RopFastTranserSourceCopyProperties**	**folderContent**	The **InputServerObject** field is a Folder object.<22>
	messageContent	The **InputServerObject** field is a Message object.

ROP that initiates an operation	Root element in the produced FastTransfer stream	ROP request buffer field conditions
	attachmentContent	The **InputServerObject** field is an Attachment object.<23>
RopFastTranserSourceCopyMessages	**messageList**	Always.
RopFastTranserSourceCopyFolder	**topFolder**	Always.

FastTransfer streams produced by operations initiated by the **RopSynchronizationConfigure** ROP are intended for processing on the client only.

FastTransfer streams produced by operations initiated with the **RopFastTransferSource*** ROPs can either be processed by the client or uploaded to the server through an operation initiated by the **RopFastTransferDestinationConfigure** ROP. For details about the applicability of FastTransfer streams to FastTransfer upload operations, see section 2.2.3.1.2.1.1.

3 Protocol Details

3.1 Common Details

The protocol details in this section contain formulas operating on sets of elements, which include the operators and special identifiers listed in the following table.

Operator or special identifier	Example	Definition
∪	A ∪ B	A union of two sets. Every element in the resulting set belongs to either A, or B, or both.
∩	A ∩ B	An intersection of two sets. Every element in the resulting set belongs to both A and B.
{ }	{A1,..., An}	A set consisting of elements A1 through An.
⊆ ⊇	B ⊆ A A ⊇ B	B is a subset of or equal to A: every element of B is also an element of A.
+=	Set += element	An instruction to include an element into a set. The Set is assigned to Set {element}.
ø	A = ø	An empty set: a set that contains no elements. Set A is asserted to be an empty set, it has no elements.
\	C =A \ B	A relative compliment: the elements belonging to A that are not in B. Set C is the relative compliment of sets A and B.

3.1.1 Abstract Data Model

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

This protocol includes the following abstract data model (ADM) types and elements:

- **Global.Handle**, as specified in [MS-OXCRPC] section 3.1.1.
- **Session context cookie**<24>, as specified in [MS-OXCMAPIHTTP] section 3.1.1.
- **Mailbox**, as specified in [MS-OXCMSG] section 3.1.1.2. Additional elements for the **Mailbox** ADM type are defined in section 3.1.1.1.

The following ADM types are defined in this section:

- **MessagingObject**, as specified in section 3.1.1.2.
- **ICSState**, as specified in section 3.1.1.3.

3.1.1.1 Per Mailbox

Mailboxes are represented by the **Mailbox** ADM type. The following ADM element is maintained by the client for each **Mailbox** ADM type:

Mailbox.MessagingObject: A Folder object, Attachment object, or Message object only.

3.1.1.2 Per Messaging Object

Messaging objects are represented by the **MessagingObject** ADM type. The following ADM elements are maintained by the client for each **MessagingObject** ADM type:

MessagingObject.Mid: An identifier for a **Mailbox.MessagingObject** ADM type that is a Message or Attachment object, as specified in section 2.2.1.2.1.

MessagingObject.FolderId: An identifier for a **Mailbox.MessagingObject** ADM type that is a Folder object, as specified in section 2.2.1.2.2.

MessagingObject.ParentFolderId: An identifier for a **Mailbox.MessagingObject** ADM type that is a Folder object containing another Folder object, as specified in section 2.2.1.2.4.

MessagingObject.ChangeNumber: An identifier for a version of a **Mailbox.MessagingObject** ADM type, as specified in section 2.2.1.2.3.

3.1.1.3 Per ICS State

ICS states are represented by the **ICSState** ADM type. Each **ICSState** ADM type represents the state of either a content synchronization or a hierarchy synchronization operation. The following abstract data elements are maintained for each **ICSState** ADM type:

ICSState.State: A state that identifies the **Mailbox.MessagingObject** ADM types that have been communicated to the client at a particular point in time. The following **ICSState.State** ADM element values identify the point in time the **ICSState.State** ADM element represents:

- **Initial**. The ICS state provided by the client at the beginning of the ICS operation. The server compares the values of the initial ICS state properties to its version, and downloads the differences.
- **Checkpoint**. The ICS state provided by the server during the ICS operation.
- **Final**. The ICS state provided by the server at the end of the ICS operation.

ICSState.SeenNormal: Contains a set of **Mailbox.MessagingObject.ChangeNumber** ADM element values that identify changes to normal messages that have been communicated to the client, as specified in section 2.2.1.1.2.

ICSState.SeenFAI: Contains a set of **Mailbox.MessagingObject.ChangeNumber** ADM element values that identify changes to FAI messages that have been communicated to the client, as specified in section 2.2.1.1.3.

ICSState.Read: Contains a set of **Mailbox.MessagingObject.ChangeNumber** ADM element values that identify the read state changes of messages that have been communicated to the client, as specified in section 2.2.1.1.4.

ICSState.IdsetGiven: Contains a set of **Mailbox.MessagingObject.Mid** or **Mailbox.MessagingObject.FolderId** ADM element values that exist on the client, as specified in section 2.2.1.1.1.

3.1.2 Timers

None.

3.1.3 Initialization

None.

3.1.4 Higher-Layer Triggered Events

None.

3.1.5 Message Processing Events and Sequencing Rules

ROPs discussed in this document are synchronous and MUST be executed in the order outlined for each operation specified in sections 3.3.4.1, 3.3.4.2, 3.3.4.2.1, 3.3.4.3.2, and 3.3.4.3.3. Otherwise, the client and server behavior remains undefined.

3.1.5.1 Isolating Download and Upload Operations

Upload and download operations are not always isolated transactions. Upload and download operations can be affected by other operations on messaging objects.

To counteract the lack of transaction isolation between ICS download operations and the rest of operations that occur on messaging objects at the same time, servers MUST guarantee that the final ICS state does not reflect the state of the server replica at the end of the operation, but instead reflects the actual differences downloaded to a client, combined with the initial ICS state.

3.1.5.2 Managing ICS State Properties

By using the ICS state properties specified in section 2.2.1.1, only differences that are relevant to a client are downloaded and the same information is only downloaded once. The ICS state is produced by the server, optionally modified by the client, and persisted exclusively on the client. The client passes the ICS state to the server immediately after configuring a synchronization context for download or upload. The server uses the ICS state and the synchronization scope, as defined during initialization of the synchronization download context, to determine the set of differences to download to the client. At the end of the synchronization operation, the client is given a new ICS state, commonly referred to as the final ICS state.

ICS state properties are not persisted on the server and are only present as data in the FastTransfer stream and in the fields of ROPs that support synchronization. The server uses the synchronization scope and ICS state to determine what differences to download to the client. For more server-specific details, see section 3.2.5.3. Ordinarily, the server modifies the ICS state properties and sends them back to the client in the FastTransfer stream or ROP responses. Another method of sending state information back to the client is checkpointing, as specified in section 3.3.5.6.

Note that for the purposes of reducing the wire size of the ICS state by enabling compacting of regions, as specified in section 3.1.5.5, and optimizing for performance of determining a set of differences to be downloaded to clients, servers can include extra IDs in **IDSETs** that represent CNSETs, as specified in section 2.2.2.4, as long as that will never affect the sets of differences that are downloaded to clients.

During the first synchronization of a synchronization scope, a client MUST send the relevant ICS state properties as zero-length byte arrays. The server assumes that the ICS state properties are zero-length byte arrays if a client fails to send them when setting up a content synchronization download operation. It is recommended that clients always send all ICS state properties that are relevant to a selected synchronization mode, defaulting them to zero-length byte arrays.

3.1.5.2.1 Sending and Receiving the PidTagIdsetGiven ICS State Property

The property tag for this property suggests that it is of type **PtypInteger32** ([MS-OXCDATA] section 2.11.1), but the data MUST be handled as **PtypBinary** ([MS-OXCDATA] section 2.11.1) data by both clients and servers. Clients and servers SHOULD send the **PidTagIdsetGiven** property (section 2.2.1.1.1) with a property tag that defines it as **PtypInteger32**; however, servers SHOULD accept this property when the property tag identifies it as **PtypInteger32** or **PtypBinary**.

This property is ignored for synchronization upload operations and is not downloaded back to the client in the final ICS state obtained for them through the **RopSynchronizationGetTransferState** ROP. Clients SHOULD remove this property before uploading the initial ICS state on synchronization upload contexts and clients MUST merge this property back in when receiving the final ICS state from the server. However, if the client does not remove this property before uploading the initial ICS state, there is no server impact. Clients MUST add IDs of messaging objects created in or originating from a local replica to this property by using a process called checkpointing, as specified in section 3.3.5.6.

3.1.5.3 Identifying Objects and Maintaining Change Numbers

On creation, objects in the mailbox are assigned internal identifiers, commonly known as **Folder ID** structures ([MS-OXCDATA] section 2.2.1.1) for folders and **Message ID** structures ([MS-OXCDATA] section 2.2.1.2) for messages. After internal identifiers are assigned to an object, they MUST never be reused, even if the object it was first assigned to no longer exists. Copying of messaging objects within a mailbox or moving messages between folders of the same mailbox translates into creation of new messaging objects and therefore, new internal identifiers MUST be assigned to new copies. All communications with the server MUST be based on these server-compatible internal identifiers. All other observed behavior is an implementation detail, and not a part of the protocol, and therefore MUST NOT be relied upon.

In most cases, the server is responsible for assigning internal identifiers to mailbox objects, which happens during execution of ROPs, such as **RopSaveChangesMessage** ([MS-OXCROPS] section 2.2.6.3) and **RopCopyTo** ([MS-OXCROPS] section 2.2.8.12), or while processing events not controlled by the client (such as Message object delivery).

Messaging objects also maintain a change number by using the **CN** structure, as specified in section 2.2.2.1, which identifies a version of an object and adheres to the same rules as internal identifiers for messaging objects. When a new object is created, it is assigned a change number. A new change number is assigned to a messaging object each time it is modified. For messages, in addition to a change number for the entire message, there are additional mechanisms for tracking changes to their elements, such as the read state, as specified in section 3.2.5.6, and properties and subobjects arranged into groups, as specified in section 3.2.5.7.

For folders, a change number indicates whether the folder itself has changed, it does not indicate whether the contained messages or aggregated folder properties have changed. If a message within a folder changes, the change number is not updated; however, the aggregated property **PidTagLocalCommitTimeMax** property ([MS-OXCFOLD] section 2.2.2.2.1.14) is modified to reflect that something within the folder has been changed. Also, if a message is deleted within the folder, the value of the folder change number does not change, but the aggregated **PidTagDeletedCountTotal** property ([MS-OXCFOLD] section 2.2.2.2.1.15) is updated to reflect the change. A client can monitor these aggregated properties on a folder to determine whether the folder has changed and whether the client needs to synchronize the contents of the folder.

A protocol role that generates internal identifiers for messaging objects and changes MUST ensure that the **GLOBCNT** structure portions, as specified in section 2.2.2.5, of the internal identifiers that share the same **REPLGUID** structure, as specified in the **XID** structure in section 2.2.2.2, only increase with time, when compared byte to byte.

Whenever a change number is changed on a messaging object as the result of the direct modification of the object in a replica (1), as opposed to synchronization, its **Predecessor Change List (PCL)** MUST be merged with the **XID** value that represents the new change number.

Clients that use ICS upload to synchronize their local replica with a server replica MUST assign identifiers to client-originated objects in a local replica by using one of the mechanisms specified in section 3.3.5.2.1. Clients MUST generate foreign identifiers, as specified in section 3.3.5.2.3, to identify client-side changes to objects that they export through ICS upload.

Upon successful import of a new or changed object using ICS upload, the server MUST do the following when receiving the **RopSaveChangesMessage** ROP:

- Assign the object a new internal change number (**PidTagChangeNumber** property (section 2.2.1.2.3)). This is necessary because the server MUST be able to represent the imported version in the **MetaTagCnsetSeen** (section 2.2.1.1.2) or **MetaTagCnsetSeenFAI** (section 2.2.1.1.3) properties, and these properties cannot operate on **foreign identifiers** for change numbers that a client passes.
- Assign the object an internal identifier. If the object is a folder, the **PidTagFolderId** property (section 2.2.1.2.2) is assigned. If the object is a message, the **PidTagMid** property (section 2.2.1.2.1) is assigned.
    - Convert the **GID** structure ([MS-OXCDATA] section 2.2.1.3) to a short-term internal identifier and assign it to an imported object, if the external identifier is a **GID** value.
- Assign the object the given **PidTagChangeKey** property value (section 2.2.1.2.7) and **PidTagPredecessorChangeList** (section 2.2.1.2.8) that equals PCL {**PidTagChangeKey**}.

If the import of the object triggered detection of a conflict, the server MUST follow the previous steps for a version of the object resulting from the conflict resolution. For details about handling conflict, see section 3.1.5.6.

Foreign identifiers supplied by clients for change identification (such as the **PidTagChangeKey** property) are replaced whenever their corresponding internal identifiers change. Examples are provided in the following table. The table uses the following notation:

- Equals sign (=) to specify that a property is set to the value specified and uses the equals and p. For example, **PidTagSourceKey** = GID(ID1) means that the **PidTagSourceKey** property (section 2.2.1.2.5) is set to the value of the initial **global identifier**.
- Plus sign followed by equals sign (+=) to specify that the value has been incremented. For example, an initial change number of 1 (CN1) increments to 2 (CN2) as changes are made to the message or folder.

Sequence of client action	Updates made on the server
RopSynchronizationImportMessageChange ROP (section 2.2.3.2.4.2) for a new message: ▪ **PidTagSourceKey** = GID(ID1) ▪ **PidTagChangeKey** = XCN1 Client checkpoints the stored initial ICS state: ▪ **MetaTagIdsetGiven** += ID2	▪ **PidTagSourceKey** = GID(ID1) ▪ **PidTagMid**= ID1 ▪ **PidTagChangeKey** = XCN1 ▪ **PidTagChangeNumber** = CN2 ▪ Final ICS state: **MetaTagCnsetSeen** += CN2
RopSynchronizationImportMessageChange ROP	▪ **PidTagChangeKey** = XCN3

Sequence of client action	Updates made on the server
▪ **PidTagSourceKey** = GID(ID1) ▪ **PidTagChangeKey** = XCN3	▪ **PidTagChangeNumber** = CN4 ▪ Final ICS state: **MetaTagCnsetSeen** += CN4
ICS download of contents	▪ **PidTagSourceKey** = GID(ID1) ▪ **PidTagMid** = ID1 ▪ **PidTagChangeKey** = XCN3 ▪ **PidTagChangeNumber** = CN4
RopOpenMessage ROP ([MS-OXCROPS] section 2.2.6.1) **RopSetProperties** ROP ([MS-OXCROPS] section 2.2.8.6) **RopSaveChangesMessage** ROP	▪ **PidTagChangeNumber** = CN5
ICS download	▪ Changes to a message: ▪ **PidTagSourceKey** = GID(ID1) ▪ **PidTagMid** = ID1 ▪ **PidTagChangeKey** = GID(CN5) ▪ **PidTagChangeNumber** = CN5 ▪ Final ICS state: **MetaTagCnsetSeen** += CN5
RopSynchronizationImportMessageMove ROP (section 2.2.3.2.4.4)	▪ Message is hard deleted in the source folder A. ▪ A copy of the message is created in destination folder B with: ▪ **PidTagMid** = ID2 ▪ **PidTagChangeNumber** = CN6
ICS download of contents for folder A	▪ Deletions: ID1 ▪ Final ICS state: **MetaTagIdsetGiven** -= ID1
ICS download of contents for folder B	▪ New message: ▪ **PidTagSourceKey** = GID(ID2) ▪ **PidTagMid** = ID2 ▪ **PidTagChangeKey** = GID(CN6) ▪ **PidTagChangeNumber** = CN6 ▪ Final ICS state:

Sequence of client action	Updates made on the server
	▪ **MetaTagIdsetGiven** -= ID2 ▪ **MetaTagCnsetSeen** += CN6
RopSynchronizationImportMessageChange ROP ▪ **PidTagSourceKey** = GID(ID2) ▪ **PidTagChangeKey** = XCN7	▪ **PidTagChangeKey** = XCN7 ▪ **PidTagChangeNumber** = CN8

3.1.5.4 Serializing an IDSET Structure

When an **IDSET** structure, as specified in section 2.2.2.4, has to be transmitted from a client to a server or from a server to a client, it has to be serialized. This section specifies details about how to serialize an **IDSET**.

3.1.5.4.1 Formatted IDSET Structures

Before serialization, the contents of an **IDSET** structure, as specified in section 2.2.2.4, have to be arranged in such a way as to allow it to be properly encoded. The ID values MUST be arranged by **REPLID** structure value and all IDs for each **REPLID** MUST be reduced into a **GLOBSET**, as specified in section 2.2.2.6, of **GLOBCNTs**, as specified in section 2.2.2.5. Each **GLOBSET** MUST be arranged from lowest to highest **GLOBCNT** value where all duplicate **GLOBCNT** values are removed. The remaining **GLOBCNT** values MUST be grouped into consecutive ranges with a low **GLOBCNT** value and a high **GLOBCNT** value. If a **GLOBCNT** value is disjoint it MUST be made into a singleton range with the low and high **GLOBCNT** values being the same. The following figure shows what a properly formatted **IDSET** looks like for serialization.

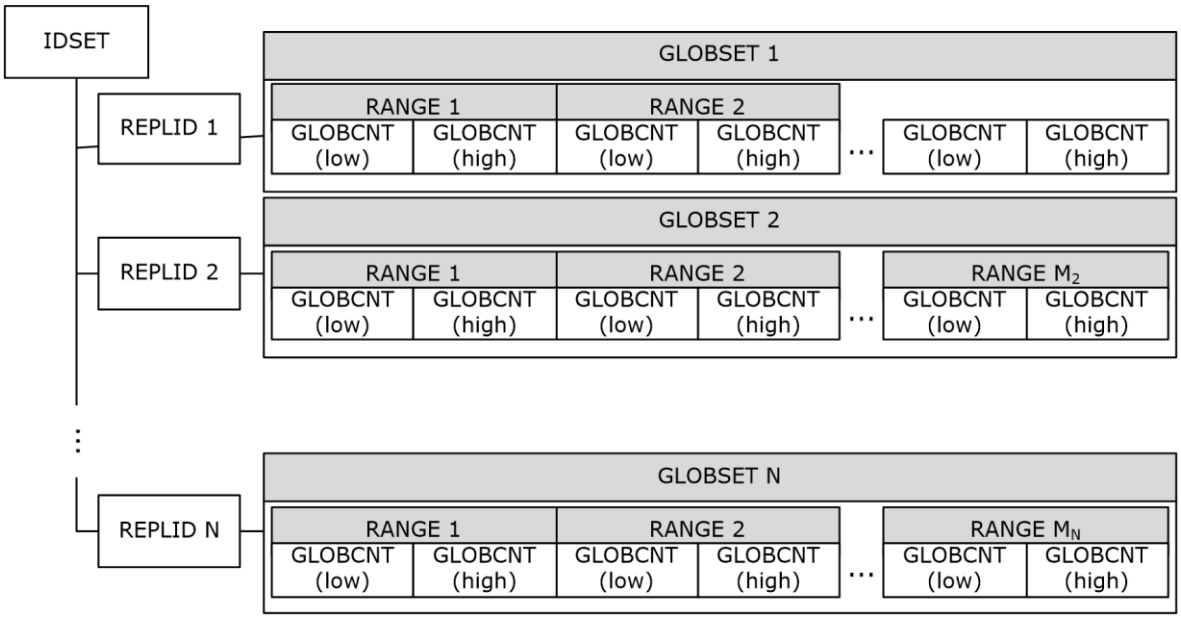


Figure 1: Formatted IDSET structure

3.1.5.4.2 IDSET Serialization

There are two different formats in which a serialized **IDSET** structure, as specified in section 2.2.2.4, can exist on the wire. The only difference is how the **REPLID** structure is represented in the serialization buffer. The first format contains the **REPLID** followed by the **GLOBSET**, as specified in section 2.2.2.6, and is used by the following properties: **MetaTagIdsetDeleted** (section 2.2.1.3.1), **MetaTagIdsetNoLongerInScope** (section 2.2.1.3.2), **MetaTagIdsetExpired** (section 2.2.1.3.3), **MetaTagIdsetRead** (section 2.2.1.3.4), and **MetaTagIdsetUnread** (section 2.2.1.3.5). The second format contains, instead of the **REPLID**, the **REPLGUID** that is associated with the **REPLID**, followed by the **GLOBSET**, and it is used by the following properties: **MetaTagIdsetGiven** (section 2.2.1.1.1), **MetaTagCnsetSeen** (section 2.2.1.1.2), **MetaTagCnsetSeenFAI** (section 2.2.1.1.3), **MetaTagCnsetRead** (section 2.2.1.1.4). No information contained in the serialized buffer identifies which format is being used. The context in which the serialized **IDSET** structure is being used on the wire dictates which format MUST be used: if an **IDSET** was persisted or is intended to be persisted across sessions, such as when it represents a portion of an ICS state, as specified in section 2.2.1.1, it MUST be transmitted in the **REPLGUID**-based form. If it's only a part of a transient set of data, like IDs of items that were deleted since the last synchronization, as specified in section 2.2.1.3.1, it MUST be transmitted in a **REPLID**-based form. Sections 3.1.5.4.3 through section 3.1.5.4.3.2.5 specify the layout of both formats on the wire. **REPLID**-based format can be converted to **REPLGUID**-based format by using mapping operations, as specified in [MS-OXCSTOR].

For more details about the format of each serialized **IDSET** structure, see section 2.2.2.4.

3.1.5.4.3 GLOBSET Serialization

The serialization of **IDSET** structures, as specified in section 3.1.5.4.2, requires each **GLOBSET** structure, as specified in section 2.2.2.6, within the **IDSET**, as specified in section 2.2.2.4, to be serialized. The **GLOBCNT** ranges, as specified in section 2.2.2.5, within the **GLOBSET** structure are serialized by using special encoding commands to compress the amount of data for each **GLOBCNT** pair. This section specifies details about how to encode and decode a **GLOBSET** during **IDSET** serialization.

Because compression is achieved by using a common byte stack to encode or decode the high order bytes of all **GLOBCNT** values, the encoder or decoder MUST construct a byte stack before implementing any of these commands.

3.1.5.4.3.1 Encoding

The commands specified in the following sections can be used to encode a **GLOBSET** structure, as specified in section 2.2.2.6.

Aside from the requirements set forth in this section, this specification does not mandate how the encoding and decoding commands are used. When more than one command can be used to achieve the same result set, the choice of command used is an implementation decision.

3.1.5.4.3.1.1 Push Command (0x01 – 0x06)

The **Push** command, as specified in section 2.2.2.6.1, SHOULD be used when multiple **GLOBCNT** structure values, as specified in section 2.2.2.5, share the same high-order values. For example, if all **GLOBCNT** values have the same two high-order bytes, use the **Push** command (0x02) to push two bytes onto the common byte stack. These two bytes are used to create **GLOBCNT** value pairs during decoding.

The **Push** command can also be used to generate an encoding for a singleton range where the low value and the high value are the same. When a **Push** command places a sixth byte onto the common byte stack, it tells the decoder the next **GLOBCNT** pair has all six bytes in common. This places a singleton **GLOBCNT** range into the **GLOBSET** structure, as specified in section 2.2.2.6, when decoded. The values added to the common byte stack on the last **Push** command are removed automatically and do not require a **Pop** command.

For more details about the format of the **Push** command, see section 2.2.2.6.1.

3.1.5.4.3.1.2 Pop Command (0x50)

Bytes that have been pushed onto the common byte stack with a **Push** command, as specified in section 2.2.2.6.1, can be removed using the **Pop** command, as specified in section 2.2.2.6.2. The **Push** and **Pop** commands are used together to adjust the bytes that are stored on the common byte stack. The common byte stack is used to reduce the amount of serialized data if the **GLOBCNT** structures, as specified in section 2.2.2.5, all share common high-order bytes. This allows for those common high-order bytes to be encoded and placed into the serialization buffer only once and not repeated with every **GLOBCNT**. The **Pop** command MUST NOT be used if no bytes are currently on the common byte stack.

For more details about the format of the **Pop** command, see section 2.2.2.6.2.

3.1.5.4.3.1.3 Bitmask Command (0x42)

The **Bitmask** command, as specified in section 2.2.2.6.3, is used when there are multiple **GLOBCNT** ranges, as specified in section 2.2.2.5, that share five high-order bytes in common and the low-order bytes are all within 8 values of each other. Each **GLOBCNT** range is represented by one or more bits in a bitmask. There MUST already be five high-order bytes in the common byte stack to use this command. The **Bitmask** command can only represent at most five **GLOBCNT** ranges.

For more details about the format of the **Bitmask** command and its fields, see section 2.2.2.6.3.

The **StartingValue** field, as specified in section 2.2.2.6.3, MUST be set to the low-order byte of the low value of the first **GLOBCNT** range. The **Bitmask** field, specified in section 2.2.2.6.3, MUST have one bit set for each value within a range, excluding the low value of the first **GLOBCNT** range. The bit to set for each value within a range is determined by subtracting the low-order byte of the **GLOBCNT** from one more than the value of the **StartingValue** field. This produces a 0 based bit number value, where zero (0) is the lowest order bit and 7 is the highest order bit in the **Bitmask** field. For all **GLOBCNTs** between ranges, the bit associated with the value is not set in the bitmask.

For example, given a set of ranges where all have the same five high-order bytes in common and the low-order bytes are the values {0x01-0x03, 0x05-0x05, 0x07-0x09}, it would be encoded with a value for the **StartingValue** field of 0x01 and the value of the **Bitmask** field would be 0xEB. The value of the **Bitmask** field is broken down in the following table.

Low-Order Byte Value	0x09	0x08	0x07	0x06	0x05	0x04	0x03	0x02
Bit Number	7	6	5	4	3	2	1	0
Bit Value	1	1	1	0	1	0	1	1

If you take the value of the **StartingValue** field and each low-order byte value corresponding to a bit that is set in the **Bitmask** field, you end up with the low-order byte values {0x01, 0x02, 0x03, 0x05, 0x07, 0x08, 0x09}. If you collapse these into ranges, you have {0x01-0x03, 0x05-0x05, 0x07-0x09}.

3.1.5.4.3.1.4 Range Command (0x52)

The **Range** command, as specified in section 2.2.2.6.4, is used to generate a single **GLOBCNT** structure range, as specified in section 2.2.2.5. If the low and high value of the **GLOBCNT** range are

not the same, or the range has values that are more than 8 bytes from each other or the low and high value do not share five high-order bytes in common, the **Range** command MUST be used.

If the low and high **GLOBCNT** values share common high-order bytes, these SHOULD be pushed onto the common byte stack by using the **Push** command, as specified in section 2.2.2.6.1, prior to using the **Range** command. The low-order bytes that are not in common are used to build the **Range** command.

For more details about the format of the **Range** command and its fields, see section 2.2.2.6.4.

3.1.5.4.3.1.5 End Command (0x00)

The **End** command, as specified in section 2.2.2.6.5, is used to signal the end of the **GLOBSET** structure encoding, as specified in section 2.2.2.6. This command MUST be added after all **GLOBCNT** structure ranges, as specified in section 2.2.2.5, within the **GLOBSET** have been encoded. The **End** command can only be used if the common byte stack is empty. If after all **GLOBCNT** ranges have been encoded, there are still bytes on the common byte stack, they MUST be removed with one or more **Pop** commands, as specified in section 2.2.2.6.1, before the **End** command can be used.

For more details about the format of the **End** command, see section 2.2.2.6.5.

3.1.5.4.3.2 Decoding

The commands specified in this section can exist in a serialized **GLOBSET** structure. The server SHOULD send the client an **RpcFormat** error (0x000004B6), or MAY send a **FormatError** error (0x000004ED)<25>, as specified in [MS-OXCDATA] section 2.4.1, if it encounters an unsupported command or any other decoding failures.

3.1.5.4.3.2.1 Push Command (0x01 – 0x06)

The **Push** command, as specified in section 2.2.2.6.1, can add 2 to 6 bytes of high-order bytes to a common byte stack. When a decoding role encounters a **Push** command during the decoding process, the decoder adds the number of bytes indicated by the **Push** command to a common byte stack from highest to lowest byte order. The common byte stack is used in conjunction with subsequent encoding commands to build **GLOBCNT** structure value pairs, as specified in section 2.2.2.5, that represent **GLOBCNT** ranges within the **GLOBSET** structure. When building a **GLOBCNT**, all the bytes on the common byte stack are used and any remaining bytes required for a complete **GLOBCNT** have to come from the next encoding command in the stream.

For more details about the format of the **Push** command in the serialization buffer, see section 2.2.2.6.1.

3.1.5.4.3.2.2 Pop Command (0x50)

The **Pop** command, as specified in section 2.2.2.6.1, removes the bytes that were previously pushed onto the common byte stack from the last **Push** command, as specified in section 2.2.2.6.1. The **Pop** command unwinds the stack in the reverse order in which the bytes where pushed.

For more details about the format of the **Pop** command in the serialization buffer, see section 2.2.2.6.2.

3.1.5.4.3.2.3 Bitmask Command (0x42)

The decoder only encounters the **Bitmask** command, as specified in section 2.2.2.6.3, when there are five bytes in the common byte stack. The server SHOULD send the client an **RpcFormat** error (0x000004B6), and MAY send the client a **FormatError** error (0x000004ED)<26>, as specified in [MS-OXCDATA] section 2.4.1, if the decoder encounters the **Bitmask** command when there are more or fewer than five bytes in the common byte stack.

For more details about the format of the **Bitmask** command and its fields, see section 2.2.2.6.3.

Using the **StartingValue** and the **Bitmask** fields, as specified in section 2.2.2.6.3, a set of low-order bytes can be produced. For more details about decoding the **Bitmask** field to produce individual low-order values, see section 3.1.5.4.3.1.3. Each low-order byte MUST be combined with the required five high-order bytes on the common byte stack to form a complete 6-byte **GLOBCNT** structure, as specified in section 2.2.2.5, which MUST be added to the **GLOBSET** structure.

3.1.5.4.3.2.4 Range Command (0x52)

The **Range** command, as specified in section 2.2.2.6.4, generates a **GLOBCNT** structure range, as specified in section 2.2.2.5. The **GLOBCNT** structure range MUST be added to the **GLOBSET** structure.

For details about the format of the **Range** command and its fields, see section 2.2.2.6.4.

The **Range** command contains two byte array fields, the **LowValue** and **HighValue** as specified in section 2.2.2.6.4. Each of these fields MUST be combined with any high-order bytes in the common byte stack to produce a 6-byte **GLOBCNT** structure. The two **GLOBCNT** structures are the low and high value of the **GLOBCNT** range.

The server SHOULD send the client an **RpcFormat** error (0x000004B6), and MAY send the client a **FormatError** error (0x000004ED)<27>, as specified in [MS-OXCDATA] section 2.4.1, if the high value of the range is larger than the low value of the range.

3.1.5.4.3.2.5 End Command (0x00)

When the **End** command, as specified in section 2.2.2.6.5, is encountered, the **GLOBSET** structure MUST be complete based on the **GLOBCNT** structure values, as specified in section 2.2.2.5, generated from any previous encoding commands.

3.1.5.5 Creating Compact IDSET Structures

As the number of changes that happen to a folder grows over time, the sets of **Message ID** ([MS-OXCDATA] section 2.2.1.2) and **CN** structures, as specified in section 2.2.2.1, that are kept in an **IDSET** structure, as specified in section 2.2.2.4, grow as well. The size of the **IDSET** is rarely a problem for hierarchy synchronization operations due to the small number of folders commonly present in mailboxes. Therefore, this discussion focuses on content synchronization operations. In this section, the term **IDSET** is used to refer to both **IDSET** and **CNSET** structures.

The following mechanisms are available to help optimize **IDSET** structures for performance:

1. **IDSET** compression: The wire format of **IDSET** structures is optimized for consecutive ranges and sets of nonconsecutive IDs that have close values.

2. Clustering of IDs: Clients and servers SHOULD allocate IDs of messages within a folder from contiguous sets of IDs. This optimization is based on an assumption that with time, all old messages are either deleted or moved to another folder, and so all of their IDs could be represented as one range. For more details, see section 3.3.5.2.1.

3. Collapsing of ranges: If an **IDSET** structure is never iterated over and is only used in operations like "not in", it is possible to add ranges of IDs to the **IDSET** structure to help collapse its regions, if that would not affect the results of operations it is used in.

   Note that because the synchronization scope limits synchronization to one folder, and the algorithm for determining the difference between replicas (1), as specified in section 3.2.5.3, only checks that a certain ID is not in the MetaTagCnset * properties, it is possible to add **CN** structures that were either never used or used on objects outside the synchronization scope to these **IDSET** structures without affecting the outcome. Note that this MUST NOT be done for **IDSET** structures that are

ever iterated over, such as the **MetaTagIdsetGiven** property (section 2.2.1.1.1), as it will change the outcome.

For example, an **IDSET** structure contains [10; 20] and [30; 40] for some **REPLGUID**. Because every internal change number within the same **REPLGUID** structure MUST be greater than any previous one, and the change numbers [21; 29] do not belong to any messages in the current folder, the two regions can be safely collapsed into [10; 40].

3.1.5.6 Conflict Handling

The properties that are associated with a message or a folder can be modified by the server or client at any time. Synchronizing these changes can result in conflicts in which a server or a client has to determine which set of message properties or folder properties to use: the local copy, or the copy being replicated.

This specification does not mandate that clients implement any **conflict handling**. However, if clients do implement conflict handling, their conflict handling logic MUST use an algorithm that provides the same output as the one mandated for servers, as specified in this section, to ensure the consistency of user experience regardless of the protocol role performing the conflict handling. When referring to synchronization in this specification, both download and upload are considered, unless specified otherwise.

3.1.5.6.1 Detection

Servers MUST implement conflict detection using an algorithm that provides the same output as the one described in this section.

Servers MUST perform conflict detection on ICS uploads for versions of messaging objects stored in a server replica and passed by the client through the **RopSynchronizationImport*** ROPs.

Conflict detection is performed by examining the **PidTagPredecessorChangeList** properties (section 2.2.1.2.8) for objects that have the same value for the **PidTagSourceKey** property (section 2.2.1.2.5).

Clients can perform conflict detection during ICS download for versions of objects stored in a local replica and passed by the server in a FastTransfer stream.

To illustrate the use of PCLs in conflict detection, the following algorithm uses sample PCLs (PCLA and PCLB) to detect a conflict between two versions of the same messaging object.

Conflict Detection Algorithm

PCLA includes PCLB if and only if for every **XID** structure, as specified in section 2.2.2.2, in PCLB there is an **XID** structure in PCLA that has a **NamespaceGuid** field with the same value, and the same or greater value for the **LocalId** field. The notation PCLA ≥ PCLB is used if PCLA includes or is equal to PCLB.

If a change to a messaging object is being synchronized from replica A to replica B, use the following statements to identify the conflict and the version to replicate:

1. If PCLA includes PCLB, then the version from replica A is newer and replaces the version in replica B.

2. If PCLB includes or is equal to PCLA, then the version from replica A is older, and is ignored. The version in replica B remains intact.

3. If neither 1 nor 2 is true, then versions from replicas A and B are in conflict.

Servers can create and implement additional conflict detection mechanisms, as long as PCLs for object versions that do and do not conflict adhere to these criteria.

The following figure shows how to detect a synchronization conflict when comparing the **PidTagPredecessorChangeList** property (section 2.2.1.2.8).

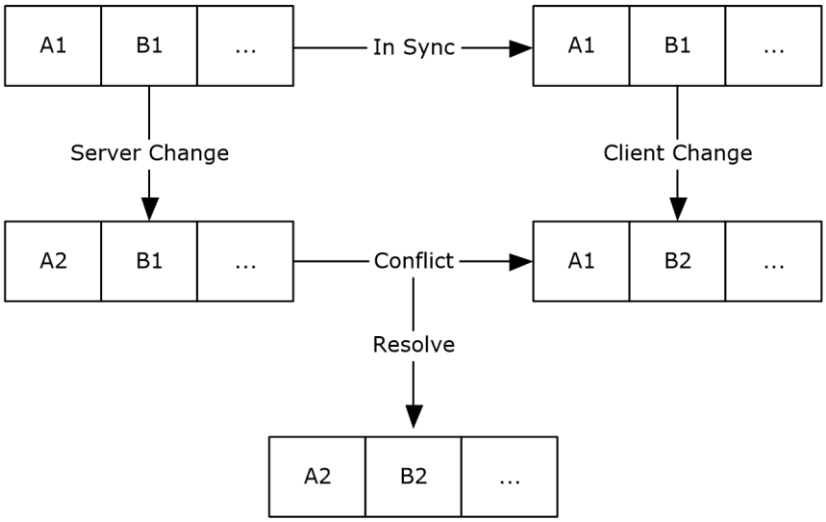


Figure 2: Conflict details

The figure simplifies the contents of the PCL to focus on the comparison. Each **CN** structure, as specified in section 2.2.2.1, within the PCL in the figure is represented by a letter (A for server changes, and B for client changes) and an increasing number.

The server-side PCL is missing a B2 change, while the client side PCL is missing A2; therefore, each has changes the other has not seen, and thus these modifications are conflicting.

The following sections describe the details of synchronization when detecting conflicting changes.

3.1.5.6.2 Resolution

At a minimum, servers MUST implement conflict resolution to the extent specified in this section. Servers can implement additional resolution algorithms. Any additional resolution algorithms MUST NOT result in the creation of conflict resolve messages, as specified in section 3.1.5.6.2.1.

A version that results from conflict resolution MUST have a PCL that makes it a successor of all conflicting versions. To achieve that, protocol roles SHOULD assign the successor a PCL created by merging the PCLs of all conflicting versions.

Version X is a successor of versions A and B if and only if the conflict detection algorithm specified in section 3.1.5.6.1 would determine that X is not in conflict and is newer than both A and B.

PCLX is a **merge** of PCLA and PCLB if and only if all of the following statements are true:

PCLX ⊆ (PCLA ∪ PCLB)

PCLX ≥ PCLA

PCLX ≥ PCLB

3.1.5.6.2.1 Conflict Resolve Message

A conflict resolve message provides a way to encapsulate conflicting versions of a Message object into a single Message object, by storing all the versions of the Message object as individual attachments to the new Message object and choosing a temporary winning message and copying it as the message contents. The contents of the conflict resolve message include all properties and subobjects of the winning version; therefore the conflict resolve message can be used in place of the winning version whenever required. The winner MUST be determined by the last writer wins algorithm, as specified in section 3.1.5.6.2.2. Because the conflict resolve message is a successor of all the conflicting versions it represents, its PCL MUST be the merge of the PCLs of the conflicting versions.

Conflict resolve messages MUST NOT be synchronized as Message objects. Instead, each attachment that represents a version in conflict MUST be synchronized as a separate Message object. The protocol role that is receiving the conflicting Message objects MUST detect the conflict during synchronization, generate a conflict resolve message locally, and resolve the conflict while considering all (possibly, more than two) conflicting versions.

A conflict resolve message MUST contain the **msInConflict** flag in the **PidTagMessageStatus** property ([MS-OXCMSG] section 2.2.1.8). Each attachment that represents an alternate replica MUST have the value of the **PidTagInConflict** property set to **TRUE**. This allows them to be distinguished from other "regular" attachments on the message.

The client and server MUST generate a conflict resolve message when detecting a conflict against the current version of a message in the replica during synchronization. It is important to understand that it is possible that the current version of the message in the local replica was transmitted during the current synchronization operation. This happens when the conflict already exists on the server before any of the conflicting messages were downloaded to the local replica.

A client or server MUST NOT generate conflict resolve messages for FAI messages. These messages MUST be resolved by using RESOLVE_METHOD_LAST_WRITER_WINS semantics, as specified in section 3.1.5.6.2.2.

3.1.5.6.2.2 Last Writer Wins Algorithm

The last writer wins algorithm uses the **PidTagLastModificationTime** ([MS-OXPROPS] section 2.755) property to determine the winning version of the folder or message, as specified in the following steps:

1. The version with the most recent **PidTagLastModificationTime** wins.

2. For messages, if the value of the **PidTagLastModificationTime** property is equal on both objects, the tie-breaking winner is determined by comparing byte-to-byte values of the **NamespaceGuid** field for **XID** structures, as specified in section 2.2.2.2, in the **PidTagChangeKey** properties (section 2.2.1.2.7). The message with the larger **NamespaceGuid** field wins. For folders, if the value of the **PidTagLastModificationTime** property is equal on both objects, the server version is kept.

3. If the byte-to-byte comparison in step 2 determines that the **NamespaceGuid** fields are equal, the version being imported wins.

The last writer wins algorithm MUST be used for conflicts detected during hierarchy synchronization and content synchronization operations on normal messages (unless the **RESOLVE_NO_CONFLICT_NOTIFICATION** flag is set in the **PidTagResolveMethod** property (section 2.2.1.4.1) set on the folder) as well as FAI messages, and folders.

3.1.5.6.3 Reporting

Conflict reporting, if deemed necessary by the value of the **PidTagResolveMethod** property (section 2.2.1.4.1) of the folder, SHOULD be done through a combination of the following methods:

1. Failing the ROP that detected the conflict.

2. Creating a conflict resolve message.

3. Creating a conflict notification message, as specified in section 3.1.5.6.3.1.

Servers MUST implement conflict reporting by failing ROPs and creating conflict resolve messages. Servers MAY implement other means of conflict reporting.

The use of the conflict resolve message combines semi-automatic conflict resolution with conflict reporting: the message has all properties of the winning version, while at the same time it contains all conflicting versions as attachments, which clients can use to offer manual conflict resolution.

Determining whether to perform conflict reporting, and what method of conflict reporting to use, is dependent on the operation that triggered the conflict detection, as specified in section 3.1.5.6.1, and on the value of the **PidTagResolveMethod** property on the folder, whose values are specified in section 2.2.1.4.1.

This controls whether the **RopSynchronizationImportMessageChange** ROP (section 2.2.3.2.4.2) is required to perform conflict reporting by failing the ROP or by creating a conflict notification message. However, the **RopSynchronizationImportHierarchyChange** ROP (section 2.2.3.2.4.2) MUST detect and resolve, and SHOULD report, possible conflicts by using a conflict notification message.

3.1.5.6.3.1 Conflict Notification Message

A conflict notification message is a special message used to notify the owner of a public folder that a conflict was resolved. This message is identified by setting the value of the **PidTagMessageClass** property ([MS-OXCMSG] section 2.2.1.3) to "IPM.Conflict.Message" which is used to notify a user that a conflict resolve note has been created. This notification MUST NOT be generated for public folder conflicts if the RESOLVE_NO_CONFLICT_NOTIFICATION flag is present in the **PidTagResolveMethod** property (section 2.2.1.4.1) for the public folder. The notification SHOULD be generated for a public folder if the RESOLVE_NO_CONFLICT_NOTIFICATION flag is not present in the **PidTagResolveMethod** property.

A conflict notification message MUST include the properties in the following table.

Property	Description
PidTagSenderName ([MS-OXOMSG] section 2.2.1.51)	Name of the folder that contains the conflict resolve message.
PidTagOriginalSubject ([MS-OXOMSG] section 2.2.2.16)	Original subject of the message.
PidTagConflictEntryId ([MS-OXPROPS] section 2.632)	EntryID of the conflict resolve message.

3.1.6 Timer Events

None.

3.1.7 Other Local Events

None.

3.2 Server Details

3.2.1 Abstract Data Model

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

This protocol includes the following ADM element:

- **Global.Handle**, as specified in [MS-OXCRPC] section 3.1.1.
- **Session context cookie**<28>, as specified in [MS-OXCMAPIHTTP] section 3.1.1.

3.2.2 Timers

None.

3.2.3 Initialization

None.

3.2.4 Higher-Layer Triggered Events

None.

3.2.5 Message Processing Events and Sequencing Rules

3.2.5.1 Isolating Download and Upload Operations

Upload and download operations are not always isolated transactions. Upload and download operations can be affected by other operations on messaging objects.

To counteract the lack of transaction isolation between ICS download operations and the rest of operations that occur on messaging objects at the same time, servers MUST confirm that the final ICS state does not reflect the state of the server replica at the end of the operation, but instead reflects the actual differences downloaded to a client, combined with the initial ICS state.

3.2.5.2 Managing the ICS State on the Server

By using the ICS state properties specified in section 2.2.1.1, the server only downloads information that is relevant to the client, and the same information is only downloaded once. The ICS state is produced by the server, and sent to the client as part of the final ICS state, but the ICS state properties are not persisted on the server.

The server receives the ICS state properties from the client using the ROPs specified in section 2.2.3.2.2 immediately after the client configures the synchronization context for download or upload. The server uses the ICS state properties and the synchronization scope, as defined during initialization of the synchronization download context, to determine the set of differences to download to the client. At the end of the synchronization operation, the server sends the client a new ICS state, commonly referred to as the final ICS state, using the **state** element, as specified in section 2.2.4.3.25. For more details about how the server determines what data to download, see section 3.2.5.3.

ICS state properties are not persisted on the server and are only present as data in the FastTransfer stream and in the fields of ROPs that support synchronization. Typically, the server modifies the ICS state properties and sends them back to the client in the FastTransfer stream or ROP responses. Another method of sending state information back to the client is client side checkpointing, as specified in section 3.3.5.6.

Note that for the purposes of reducing the wire size of the ICS state by enabling compacting of regions, as specified in section 3.1.5.5, and optimizing for performance of determining a set of differences to be downloaded to clients, servers can include extra IDs in **IDSET** structures that represent **CNSET** structures, as specified in section 2.2.2.4, as long as that never affects the sets of differences that are downloaded to clients.

3.2.5.2.1 Receiving the MetaTagIdsetGiven ICS State Property

The property tag for this property suggests that it is of type **PtypInteger32** ([MS-OXCDATA] section 2.11.1), but the data MUST be handled as **PtypBinary** ([MS-OXCDATA] section 2.11.1) data by servers. Servers SHOULD send the **MetaTagIdsetGiven** property (section 2.2.1.1.1) with a property tag that defines it as **PtypInteger32**; however, servers SHOULD accept this property when the property tag identifies it as **PtypInteger32** or **PtypBinary**.

The server ignores this property in synchronization upload operations and does not download it back to the client in the final ICS state obtained for them through the **RopSynchronizationGetTransferState** ROP (section 2.2.3.2.3.1).If the client failed to remove this property before uploading the initial ICS state, there is no effect on the server.

3.2.5.3 Determining What Differences To Download

In this section, all references to the ICS state properties refer to values uploaded in the initial ICS state.

For every object in the synchronization scope, servers MUST do the following:

- Include the following syntactical elements in the FastTransfer stream of the **OutputServerObject** field of the FastTransfer download ROPs, as specified in section 2.2.3.1.1, if one of the following applies:
  - Include the **folderChange** element, as specified in section 2.2.4.3.5, if the object specified by the **InputServerObject** field of the FastTransfer download ROP request is a Folder object
    - And the change number is not included in the value of the **MetaTagCnsetSeen** property (section 2.2.1.1.2).
  - Include the **messageChange** element, as specified in section 2.2.4.3.11, if the object specified by the **InputServerObject** field is a normal message
    - And the **Normal** flag of the **SynchronizationFlags** field was set, as specified in section 2.2.3.2.1.1.1
    - And the change number is not included in the value of the **MetaTagCnsetSeen** property.
  - Include the **messageChangeFull** element, as specified in section 2.2.4.3.13, if the object specified by the **InputServerObject** field is an FAI message, meaning the **PidTagAssociated** property (section 2.2.1.5) is set to **TRUE**
    - And the **FAI** flag of the **SynchronizationFlags** field was set
    - And the change number is not included in the value of the **MetaTagCnsetSeenFAI** property (section 2.2.1.1.3).

- If the **NoDeletions** flag of the **SynchronizationFlags** field is not set, include the **deletions** element, as specified in section 2.2.4.3.3, for objects that either:
  - Have their internal identifiers present in the value of the **MetaTagIdsetGiven** property (section 2.2.1.1.1) and are missing from the server replica.
  - Are folders that have never been reported as deleted folders.
- If the **NoDeletion** and **IgnoreNoLongerInScope** flags are not set in the **SynchronizationFlags** field, include the **deletions** element for messages that went out of scope that:
  - Have their internal identifiers present in the value of the **MetaTagIdsetGiven** property
    - And exist in a server replica and belong to a folder that defines the synchronization scope
    - And do not match the restriction that defines the synchronization scope.
- If the **ReadState** flag of the **SynchronizationFlags** field is set, include the **readStateChanges** element, as specified in section 2.2.4.3.22, for messages that:
  - Do not have their change numbers for read and unread state in the **MetaTagCnsetRead** property (section 2.2.1.1.4)
    - And are not FAI messages and have not had change information downloaded for them in this session.

The server MAY<29> confirm that the FastTransfer context that is returned by the **RopSynchronizationGetTransferState** ROP (section 2.2.3.2.3.1), which is sent before the subsequent **RopFastTransferSourceGetBuffer** ROP (section 2.2.3.1.1.5), contains only the differences that have been downloaded to the client in the current synchronization download operation, in addition to what was reflected in the initial ICS state. Note that the final ICS state that has to be downloaded in the FastTransfer stream as the last portion of the payload is exactly the same as the checkpoint ICS state that corresponds to the end of the operation.

The following invariants define the relationship between the initial ICS state, the checkpoint ICS state, and differences downloaded at the time of checkpointing. The server does not maintain a per-client state or store the values of these ICS state properties, but it does include the final ICS state at the end of the FastTransfer stream. The server does not persist the ICS state properties on the server; they are only present as data in the FastTransfer stream and in the fields of ROPs that support synchronization. The following table contains the nomenclature used to describe the invariants. For more details about checkpointing, see section 3.3.5.6.

Nomenclature	Description
Prop$_{Index}$	Property **Prop** of the ICS state, as specified in section 2.2.1.1. **Index** can be I for initial and C for checkpoint.
Prop$_{D}$	Property **Prop** that contains a particular set of differences that have been downloaded in the current operation, as specified in section 2.2.1.3.
{change$_{Subset}$**.Id}** **{change**$_{Subset}$**.CN}**	Internal identifiers (Id) or change numbers of all changes that have been downloaded in the current operation. The **Subset** can be one of the following: ▪ Omitted to denote all changes. ▪ **Normal** for normal messages. ▪ FAI for FAI messages. ▪ **Partial** for normal messages downloaded as partial changes.

Nomenclature	Description
{readStateChange.Id} **{readStateChange.ReadStateCn}**	Internal identifiers or read state change numbers of all normal messages, with only the read state changed, which have been downloaded in the current operation.

Servers MUST ensure that the following invariants are true:

- $AllDeleted = (IdsetDeleted_D \cup IdsetNoLongerInScope_D \cup IdsetExpired_D)$
- $IdsetGiven_C = (IdsetGiven_I \cup \{change.Id\}) \setminus AllDeleted$
- $CnsetSeen_C = CnsetSeen_I \cup \{change_{Normal}.Cn\}$
- $CnsetSeenFAI_C = CnsetSeenFAI_I \cup \{change_{FAI}.Cn\}$
- $CnsetRead_C = CnsetRead_I \cup \{readStateChange.ReadCn\}$
- $IdsetGiven_I \supseteq \{changes_{Partial}.Id\}$
- $IdsetGiven_I \supseteq (IdsetRead_D \cup IdsetUnread_D)$
- $\{readStateChange.Id\} = IdsetRead_D \cup IdsetUnread_D$
- $\{change.Id\} \cap AllDeleted = \emptyset$
- $\{change.Cn\} \cap (CnsetSeen_I \cup CnsetSeenFAI_I) = \emptyset$
- $\{readStateChange.Id\} \cap AllDeleted = \emptyset$
- $\{readStateChange.Id\} \cap \{change.Id\} = \emptyset$

3.2.5.4 Calculating the PidTagMessageSize Property Value

The server SHOULD make the best effort to calculate this property, but because values for properties can change before the client downloads the message, and because the client specifies what data it does and does not require, it MUST be used only as an estimate by the client.

3.2.5.5 Generating the PidTagSourceKey Value

When requested by the client, the server MUST output the **PidTagSourceKey** property (section 2.2.1.2.5) value if it is persisted, or generate it if it is missing. If the value of the **PidTagSourceKey** property is missing, the server MUST generate it by producing a **GID** structure ([MS-OXCDATA] section 2.2.1.3) from the internal identifier (**Message ID** structure ([MS-OXCDATA] section 2.2.1.2) or **Folder ID** structure ([MS-OXCDATA] section 2.2.1.1)) of the object by using the same mapping algorithm as described for the **RopLongTermIdFromId** ROP ([MS-OXCROPS] section 2.2.3.8).

The only exception is when a server is required to generate this property for a folder, which is a root of the current hierarchy synchronization download operation (that is, it is the folder that was passed to the **RopSynchronizationConfigure** ROP (section 2.2.3.2.1.1)). In this case, the **PidTagSourceKey** property MUST be output as a zero-length **PtypBinary**, as specified in [MS-OXCDATA] section 2.11.1.

3.2.5.6 Tracking Read State Changes

To conserve the bandwidth between clients and servers, the read state of the messages SHOULD be tracked separately from other changes.

Whenever the read state of a message changes on the server, a separate change number (the read state change number) on the message SHOULD be assigned a new value on the server. The change number of the message SHOULD NOT be modified unless other changes to a message were made at the same time. This allows the read state change to be efficiently downloaded to a client as an **Message ID** structure ([MS-OXCDATA] section 2.2.1.2) in the **MetaTagIdsetRead** property (section 2.2.1.3.4) **IDSET** structure or the **MetaTagIdsetUnread** property (section 2.2.1.3.5), compressed together with read state changes to other messages in the synchronization scope. An individual read state change number is never sent across the wire independently. An **IDSET** structure of change numbers associated with message read state transitions, either from read to unread, or unread to read (as determined by the **PidTagMessageFlags** property in [MS-OXCMSG] section 2.2.1.6) are included in the **MetaTagCnsetRead** property (section 2.2.1.1.4), which is part of the ICS state and is never directly set on any objects.

3.2.5.7 Working with Property Groups and Partial Changes

Property groups are defined by the **PropertyGroup** structure, as specified in section 2.2.2.8.1. The **PropertyGroup** structure contains an array of properties, referred to as a property group. One or more **PropertyGroup** structures are contained in each **PropertyGroupInfo** structure, as specified in section 2.2.2.8. Each **PropertyGroupInfo** structure is referred to as a property group mapping, as it maps the properties in the messaging object to a collection of property groups. The property group mapping is included in the FastTransfer stream in the **groupInfo** element, as specified in section 2.2.4.3.8. The **MetaTagIncrSyncGroupId** meta-property (section 2.2.4.1.5.4) is used to identify the property group mapping used on a particular messaging object, and the **MetaTagIncrementalSyncMessagePartial** meta-property (section 2.2.4.1.5.5) informs the client of the property group mapping to use when interpreting the partial item data that follows the meta-property in the FastTransfer stream.

ICS is optimized for reporting partial changes to messages using these property groups. The simplest approach for servers to provide information about partial changes is to track changes made within groups of properties. A group is considered changed if any of the properties in the group are modified or deleted. It is up to the server to define a property group mapping by adding properties to a **PropertyGroup** structure. ICS offers a way to communicate property group mapping information per-message, so every message can use its own property group mapping. However, to minimize overhead, it is recommended that the number of different mappings is kept to a minimum.

For example, a change to any property in the group of server-defined properties that track changes to message attachments would mean that all the properties in that property group are updated during the next synchronization. Likewise, a change to any property in the group of server defined properties that track changes to the body of the message would mean that all the properties in that property group are updated during the next synchronization.

To track changes to property groups on a message, servers SHOULD keep change numbers for each property group, and assign a new change number to both the group and the message whenever a change is made to a property that belongs to the group. Note that marking a message as read or unread is the most common type of message modification, and there is a specific mechanism to support just that change, as specified in section 3.2.5.6.

One message in a mailbox can have a different property group mapping than another message, which means that the properties in group N on one message can be different than the properties in group N in another message. Property group mappings do not change frequently, but they do change with server upgrades. When a message is modified and the default mapping has changed after an upgrade, the property group mapping of the message is updated.

Servers that are implemented to support<30> partial message change synchronization MUST either use a mechanism described in this section, or use an alternative mechanism that localizes changes to a message to a set of properties and subobjects, which can be unambiguously expressed by using the **messageChangePartial** element, as specified in section 2.2.4.3.15, of the FastTransfer stream. Servers that are not implemented to support partial message change synchronization ignore the

PartialItem flat of the **SendOptions** field, as specified in section 2.2.3.1.1.1.2, and download the item as a full item by using the **messageChangeFull** element, as specified in section 2.2.4.3.13, of the FastTransfer stream.

3.2.5.8 Receiving FastTransfer ROPs

3.2.5.8.1 Download

When producing FastTransfer streams for operations configured with **RopFastTransferSourceCopy*** ROPs, servers SHOULD skip over objects that the client does not have adequate permissions for. For example, if the **Move** flag of the **CopyFlags** field, as specified in section 2.2.3.1.1.1.1, is set, an additional permission to delete an object is required for the object to be included in the output FastTransfer stream. If a permission check for an object fails, the **MetaTagEcWarning** meta-property (section 2.2.4.1.5.2) SHOULD be output in a FastTransfer stream, wherever allowed by its syntactical structure, to signal a client about incomplete content.

3.2.5.8.1.1 Receiving a RopFastTransferSourceCopyTo ROP Request

When the client sends the server a **RopFastTransferSourceCopyTo** ROP request (section 2.2.3.1.1.1), the server MUST parse the request, as specified in [MS-OXCROPS] section 2.2.13.6.1 and section 2.2.3.1.1.1 of this specification. The server MUST respond with a **RopFastTransferSourceCopyTo** ROP response, as specified in [MS-OXCROPS] section 2.2.13.6.2 and section 2.2.3.1.1.1 of this specification.

If the **Level** field is set to 0x00, the server MUST copy descendant subobjects by using the property list specified by the **PropertyTags** field. Subobjects are only copied when they are not listed in the value of the **PropertyTags** field.

If the **Level** field is set to a nonzero value, the server MUST exclude all descendant subobjects from being copied.

If the **Move** flag of the **CopyFlags** field is set, the server SHOULD NOT output any objects in a FastTransfer stream that the client does not have permissions to delete.

The server MAY<31> support the **Move** flag of the **CopyFlags** field, or alternatively the server can set the value of the **ReturnValue** field to **InvalidParameter** (0x80070057) if it receives this flag.

If the **BestBody** flag of the **CopyFlags** field is set, the server SHOULD output the message body, and the body of the Embedded Message object, in their original format. If this flag is not set, the server MAY<32> output the message body in the compressed Rich Text Format (RTF).

Servers SHOULD fail the ROP if unknown flags in the **CopyFlags** field are set.

The following table lists the server behavior for valid combinations of the **Unicode**, **ForceUnicode**, and **UseCpid** flags of the **SendOptions** field.

Flag name	Description
None of the three flags are set	String properties MUST be output in the code page set on the current connection with a property type of **PtypString8** ([MS-OXCDATA] section 2.11.1).
Unicode only	String properties MUST be output either in Unicode with a property type of **PtypUnicode** ([MS-OXCDATA] section 2.11.1), or in the code page set on the current connection with a property type of **PtypString8**. If the properties are stored in Unicode on the server, the server MUST return the properties in Unicode. If the properties are not stored in Unicode on the server, the server MUST return the properties in the code page set on the current connection.
ForceUnicode only	String properties MUST be output in Unicode with a property type of **PtypUnicode**.

Flag name	Description
UseCpid only	String properties MUST be output using code page property types, as specified in section 2.2.4.1.1.1. If the properties are stored in Unicode on the server, the server MUST return the properties using the Unicode code page (code page property type 0x84B0), otherwise the server MUST send the string using the code page property type of the code page in which the property is stored on the server.
Unicode and **ForceUnicode**	String properties MUST be output in Unicode with a property type of **PtypUnicode**.
UseCpid and **Unicode**	String properties MUST be output using code page property types, as specified in section 2.2.4.1.1.1. If the properties are stored in Unicode on the server, the server MUST return the properties using the Unicode code page (code page property type 0x84B0); otherwise the server MUST send the string using the code page property type of the code page in which the property is stored on the server. The combination of the **UseCpid** and **Unicode** flags is the **ForUpload** flag.
UseCpid and **ForceUnicode**	String properties MUST be output using the Unicode code page (code page property type 0x84B0).
UseCpid, **Unicode**, and **ForceUnicode**	The combination of the **UseCpid** and **Unicode** flags is the **ForUpload** flag. String properties MUST be output using the Unicode code page (code page property type 0x84B0).

The server MUST attempt to recover from failures when downloading changes for individual messages, when the **RecoverMode** flag of the **SendOptions** field is set.

Servers SHOULD fail the ROP if any unknown flags in the **SendOptions** field are set.

3.2.5.8.1.2 Receiving a RopFastTransferSourceCopyProperties ROP Request

When the client sends the server a **RopFastTransferSourceCopyProperties** ROP (section 2.2.3.1.1.2) request, the server MUST parse the request as specified in [MS-OXCROPS] section 2.2.12.7.1 and section 2.2.3.1.1.2 of this specification. The server MUST respond with a **RopFastTransferSourceCopyProperties** ROP response, as specified in [MS-OXCROPS] section 2.2.12.7.2 and section 2.2.3.1.1.2 of this specification.

If the **Level** field is set to 0x00, the server MUST copy descendant subobjects by using the property list specified by the **PropertyTags** field. Subobjects are not copied unless listed in the value of the **PropertyTags** field.

If the **Level** field is set to a nonzero value, the server MUST exclude all descendant subobjects from being copied.

If the **Move** flag of the **CopyFlags** field is specified for a download operation, the server SHOULD NOT output any objects in a FastTransfer stream that the client does not have permissions to delete.

Servers SHOULD fail the ROP if unknown flags in the **CopyFlag** field are set.

For details about server behavior related to the **Unicode**, **ForceUnicode**, and **UseCpid** flags of the **SendOptions** field, see section 3.2.5.8.1.1.

Servers SHOULD fail the ROP if any unknown flags in the **SendOptions** field are set.

3.2.5.8.1.3 Receiving a RopFastTransferSourceCopyMessages ROP Request

When the client sends the server a **RopFastTransferSourceCopyMessages** ROP (section 2.2.3.1.1.3) request, the server MUST parse the request, as specified in [MS-OXCROPS] section 2.2.12.5.1 and section 2.2.3.1.1.3 of this specification. The server MUST respond with a

RopFastTransferSourceCopyMessages ROP response, as specified in [MS-OXCROPS] section 2.2.12.5.2 and section 2.2.3.1.1.3 of this specification.

If the **Move** flag of the **CopyFlags** field is set for a download operation, the server SHOULD NOT output any objects in a FastTransfer stream that the client does not have permissions to delete.

If the **BestBody** flag of the **CopyFlags** field is set, the server SHOULD output the message body, and the body of the Embedded Message object, in their original format. The original format of the message is determined by using the best body algorithm, as specified in [MS-OXBBODY] section 2.1.3.1.

If the **BestBody** flag of the **CopyFlags** field is not set, the server MAY<33> output message bodies in the compressed RTF.

If the **SendEntryId** flag of the **CopyFlags** field is set, the server does not remove message and change identification information from the output.

If the **SendEntryId** flag of the **CopyFlags** field is not set, the server removes message and change identification information from the output.

For details about server behavior related to the **Unicode**, **ForceUnicode**, and **UseCpid** flags of the **SendOptions** field, see section 3.2.5.8.1.1.

Servers SHOULD fail the ROP if any unknown flags in the **SendOptions** field are set.

3.2.5.8.1.4 Receiving a RopFastTransferSourceCopyFolder ROP Request

When the client sends the server a **RopFastTransferSourceCopyFolder** ROP (section 2.2.3.1.1.4) request, the server MUST parse the request, as specified in [MS-OXCROPS] section 2.2.12.4.1 and section 2.2.3.1.1.4 of this specification. The server MUST respond with a **RopFastTransferSourceCopyFolder** ROP response, as specified in [MS-OXCROPS] section 2.2.12.4.2 and section 2.2.3.1.1.4 of this specification.

If the **CopySubfolders** flag of the **CopyFlags** field is set, the server MUST recursively include the subfolders of the folder specified in the **InputServerObject** field in the scope.

If the **Move** flag of the **CopyFlags** field is set and the **CopySubfolders** flag is not set, the server MUST recursively include the subfolders of the folder specified in the **InputServerObject** field in the scope.

If the **Move** flag of the **CopyFlags** field is not set and the **CopySubfolders** flag is not set, the server MUST NOT recursively include the subfolders of the folder specified in the **InputServerObject** field in the scope.

Servers SHOULD fail the ROP if unknown flags on the **CopyFlags** field are set.

For details about server behavior related to the **Unicode**, **ForceUnicode**, and **UseCpid** flags of the **SendOptions** field, see section 3.2.5.8.1.1.

Servers SHOULD fail the ROP if any unknown flags in the **SendOptions** field are set.

3.2.5.8.1.5 Receiving a RopFastTransferSourceGetBuffer ROP Request

When the client sends the server a **RopFastTransferSourceGetBuffer** ROP (section 2.2.3.1.1.5) request, the server MUST parse the request as specified in [MS-OXCROPS] section 2.2.12.6.1 and section 2.2.3.1.1.1 of this specification. The server MUST respond with a **RopFastTransferSourceGetBuffer** ROP response, as specified in [MS-OXCROPS] section 2.2.12.6.2 and section 2.2.3.1.1.1 of this specification.

If the value of the **BufferSize** field in the ROP request is 0xBABE, the server determines the buffer size based on the residual size of the **rgbOut** field of the **EcDoRpcExt2** method, as specified in [MS-

OXCRPC] section 3.1.4.2, or the **rgbRopOut** field of the **Execute** request type response<34>, as specified in [MS-OXCMAPIHTTP] section 2.2.4.2.2.

If the value of the **BufferSize** field in the ROP request is set to 0xBABE, the server MUST limit the amount of data returned in **TransferBuffer** field to the residual size of the output buffer minus result structure overhead, or limit the amount of data returned in the **TransferBuffer** field to **MaximumBufferSize**, whichever is smaller.

If the value of **BufferSize** field in the ROP request is set to a value other than 0xBABE, the following semantics apply:

- The server MUST fail the command before processing the ROP by doing the following:
- The server MUST return the **RopBufferTooSmall** ROP ([MS-OXCROPS] section 2.2.15.1) if the resulting **BufferSize** bytes in the **TransferBuffer** field are larger than the residual **rgbOut** field of the **EcDoRpcExt2** method, as specified in [MS-OXCRPC] section 3.1.4.2, or the **rgbRopOut** field of the **Execute** request type response<35>, as specified in [MS-OXCMAPIHTTP] section 2.2.4.2.2.
- The server MUST output, at most, the number of bytes specified by the **BufferSize** field in the **TransferBuffer** field even if more data is available.
- The server returns less bytes than the value specified by the **BufferSize** field, or the server returns the number of bytes specified by the **BufferSize** field in the **TransferBuffer** field.

The **ReturnValue** field is set to 0x00000480 only when the client is version 11.0.0.4920 or higher. For more details about version checking, see [MS-OXCRPC] section 3.2.4.1.3.

Servers SHOULD fail any successive calls to the **RopFastTransferSourceGetBuffer** ROP if the previous iteration returns a value other than of **Success** or 0x00000480 in the **ReturnValue** field. Successive calls should fail with the same error as previous failed operations. The server MUST serialize each portion of the FastTransfer stream using the syntax specified in section 2.2.4 and output it using the **TransferBuffer** field.

3.2.5.8.1.6 Receiving a RopTellVersion ROP Request

When the client sends the server a **RopTellVersion** ROP (section 2.2.3.1.1.6) request, the server MUST parse the request, as specified in [MS-OXCROPS] section 2.2.12.8.1 and section 2.2.3.1.1.6 of this specification. The server MUST respond with a **RopTellVersion** ROP response, as specified in [MS-OXCROPS] section 2.2.12.8.2 and section 2.2.3.1.1.6 of this specification.

3.2.5.8.2 Upload

3.2.5.8.2.1 Receiving a RopFastTransferDestinationConfigure ROP Request

When the client sends the server a **RopFastTransferDestinationConfigure** ROP (section 2.2.3.1.2.1) request, the server MUST parse the request, as specified in [MS-OXCROPS] section 2.2.12.1.1 and section 2.2.3.1.2.1 of this specification. The server MUST respond with a **RopFastTransferDestinationConfigure** ROP response, as specified in [MS-OXCROPS] section 2.2.12.1.2 and section 2.2.3.1.2.1 of this specification.

Any changes to an object identified by **InputServerObject** in the ROP request are not persisted until the **RopSaveChangesMessage** ROP ([MS-OXCROPS] section 2.2.6.3) is called.

The server MUST stop execution of the ROP if the value of the **SourceOperation** field is unknown.

The server SHOULD<36> fail the ROP if unknown flags in the **CopyFlags** field are set, or the server MAY<37> ignore the unknown value of the **CopyFlags** field.

3.2.5.8.2.2 Receiving a RopFastTransferDestinationPutBuffer ROP Request

When the client sends the server a **RopFastTransferDestinationPutBuffer** ROP ([MS-OXCROPS] section 2.2.12.2) request, the server MUST parse the request, as specified in [MS-OXCROPS] section 2.2.12.2.1 and section 2.2.3.1.2.2 of this specification. The server MUST respond with a **RopFastTransferDestinationPutBuffer** ROP response, as specified in [MS-OXCROPS] section 2.2.12.2.2 and section 2.2.3.1.2.2 of this specification.

3.2.5.9 Receiving Incremental Change Synchronization ROPs

3.2.5.9.1 Download

3.2.5.9.1.1 Receiving a RopSynchronizationConfigure ROP Request

When the client sends the server a **RopSynchronizationConfigure** ROP (section 2.2.3.2.1.1) request, the server MUST parse the request, as specified in [MS-OXCROPS] section 2.2.13.1.1 and section 2.2.3.2.1.1 of this specification. The server MUST respond with a **RopSynchronizationConfigure** ROP response, as specified in [MS-OXCROPS] section 2.2.13.1.2 and section 2.2.3.2.1.1 of this specification.

SynchronizationType Constraints

The following constraints apply to the **SynchronizationType** field.

Servers MUST fail the ROP if the value of the **SynchronizationType** field is unknown.

If the **Unicode** flag of the **SynchronizationFlags** field is set, the client supports Unicode and the server MUST write the values of the string properties to the FastTransfer stream as they are stored. The **Unicode** flag does not specify that the server writes the value in Unicode format. The server writes the values in Unicode or non-Unicode format depending on how they are stored.

If the **Unicode** flag of the **SynchronizationFlags** field is not set, the client does not support Unicode and the server MUST write the values of string properties to the FastTransfer stream in the code page set on connection.

If the **NoDeletions** flag of the **SynchronizationFlags** field is set, the server MUST NOT download information about item deletions, as specified in section 2.2.4.3.3, and the server MUST respond as if the **IgnoreNoLongerInScope** flag was set.

If the **NoDeletions** flag of the **SynchronizationFlags** field is not set, the server MUST download information about item deletions, as specified in section 2.2.4.3.3.

If the **IgnoreNoLongerInScope** flag of the **SynchronizationFlags** field is set, the server MUST NOT download information about messages that went out of scope as deletions, as specified in section 2.2.4.3.3.

If the **IgnoreNoLongerInScope** flag of the **SynchronizationFlags** field is not set, the server MUST download information about messages that went out of scope as deletions, as specified in section 2.2.4.3.3.

If the **ReadState** flag of the **SynchronizationFlags** field is set, the server MUST also download information about changes to the read state of messages, as specified in section 2.2.4.3.22.

If the **ReadState** flag of the **SynchronizationFlags** field is not set, the server MUST NOT download information about changes to the read state of messages, as specified in section 2.2.4.3.22.

If the **FAI** flag of the **SynchronizationFlags** field is set, the server MUST download information about changes to FAI messages, as specified by the **folderContents** element in section 2.2.4.3.7.

If the **FAI** flag of the **SynchronizationFlags** field is not set, the server MUST NOT download information about changes to FAI messages, as specified by the **folderContents** element in section 2.2.4.3.7.

If the **Normal** flag of the **SynchronizationFlags** flag is set, the server MUST download information about changes to normal messages, as specified in section 2.2.4.3.11.

If the **Normal** flag of the **SynchronizationFlags** field is not set, the server MUST NOT download information about changes to normal messages, as specified in section 2.2.4.3.11.

If the **OnlySpecifiedProperties** flag of the **SynchronizationFlags** field is set, the server SHOULD limit properties and subobjects written to the FastTransfer stream for **top-level messages** to the properties listed in the **PropertyTags** field.

If the **OnlySpecifiedProperties** flag of the **SynchronizationFlags** field is not set, the server SHOULD exclude properties and subobjects from the FastTransfer stream for folders and top-level messages, if they are listed in the **PropertyTags** field.

If the **NoForeignIdentifiers** flag of the **SynchronizationFlags** field is set, the server MUST ignore any persisted values for the **PidTagSourceKey** property (section 2.2.1.2.5) and **PidTagParentSourceKey** (section 2.2.1.2.6) properties when producing the FastTransfer stream for folder and message changes.

If the **NoForeignIdentifiers** flag of the **SynchronizationFlags** field is not set, the server MUST NOT ignore any persisted values for the **PidTagSourceKey** and **PidTagParentSourceKey** properties when producing the FastTransfer stream for folder and message changes.

The server MUST fail the ROP request if the **Reserved** flag of the **SynchronizationFlags** field is set.

If the **BestBody** flag of the **SynchronizationFlags** field is set, the server SHOULD write message bodies to the FastTransfer stream in their original format. An exception is embedded messages, the server MAY honor this flag for embedded messages.

If the **BestBody** flag of the **SynchronizationFlags** field is not set, the server MAY<38> write message bodies to the FastTransfer stream in the compressed RTF format.

The **IgnoreSpecifiedOnFAI** flag is only used in conjunction with the **OnlySpecifiedProperties** flag being set. Both flags are defined as part of the **SynchronizationFlags** field. If the **OnlySpecifiedProperties** flag is not set, the server MUST ignore the **IgnoreSpecifiedOnFAI** flag.

If the **OnlySpecifiedProperties** flag is set and the **IgnoreSpecifiedOnFAI** flag is not set, the server writes only the specified properties and subobjects for all messages to the FastTransfer stream.

If the **OnlySpecifiedProperties** flag is set and the **IgnoreSpecifiedOnFAI** flag is also set, the server MUST write only the specified properties and subobjects for all non-FAI messages to the FastTransfer stream. For FAI messages, the server MUST write all properties and subobjects to the FastTransfer stream.

If the **Progress** flag of the **SynchronizationFlags** field is set, the server SHOULD inject the **progressTotal** element, as specified in section 2.2.4.3.19, into the FastTransfer stream.

If the **Progress** flag of the **SynchronizationFlags** field is not set, the server MUST not inject the **progressTotal** element into the FastTransfer stream.

Servers SHOULD fail the ROP if unknown flag bits are set, or MAY fail the ROP if additional flags, used only in server-to-server communications are set.

SynchronizationExtraFlags Constraints

The following constraints apply to the **SynchronizationExtraFlags** field, as specified in section 2.2.3.2.1.1.1.

The server MUST include either the **PidTagFolderId** property (section 2.2.1.2.2) or the **PidTagMid** property (section 2.2.1.2.1) in a folder change or message change header if and only if the **Eid** flag of the **SynchronizationExtraFlags** field is set. The server MUST include the **PidTagFolderId** property in the folder change header if the **SynchronizationType** field is set to **Hierarchy** (0x02), as specified in section 2.2.3.2.1.1.1. The server MUST include the **PidTagMid** property in the message change header if the **SynchronizationType** field is set **Contents** (0x01), as specified in section 2.2.3.2.1.1.1.

The server MUST include the **PidTagMessageSize** property (section 2.2.1.6) in the message change header if and only if the **MessageSize** flag of the **SynchronizationExtraFlags** field is set.

The server MUST include the **PidTagChangeNumber** property (section 2.2.1.2.3) in the message change header if and only if the **CN** flag of the **SynchronizationExtraFlags** field is set.

The server MUST NOT include the **PidTagChangeNumber** property in the message change header if and only if the **CN** flag of the **SynchronizationExtraFlags** field is not set.

If the **OrderByDeliveryTime** flag of the **SynchronizationExtraFlags** field is set, the server MUST sort messages by the value of their **PidTagMessageDeliveryTime** property ([MS-OXOMSG] section 2.2.3.9), or by the **PidTagLastModificationTime** property ([MS-OXPROPS] section 2.755) if the former is missing, when generating a sequence of **messageChange** elements for the FastTransfer stream, as specified in section 2.2.4.2. The server adds messages in the FastTransfer stream from newest to oldest, so the client receives the newest messages first.

If the **OrderByDeliveryTime** flag of the **SynchronizationExtraFlags** field is not set, there is no requirement on the server to return items in a specific order.

Servers MUST ignore any unknown flags in the **SynchronizationExtraFlags** field.

PropertyTags Constraints

The following constraints apply to the **PropertyTags** field.

This field has different semantics, depending on the value of the **OnlySpecifiedProperties** flag of the **SynchronizationFlags** field, as follows:

- If the **OnlySpecifiedProperties** flag is not set, the server SHOULD exclude properties and subobjects from the FastTransfer stream for folders and top-level messages, if the property is listed in the **PropertyTags** field.
- If the **OnlySpecifiedProperties** flag is set, the server SHOULD limit properties and subobjects written to the FastTransfer stream for top-level messages to properties listed in the **PropertyTags** field.

Inclusion of properties that denote message subobjects in the **PropertyTags** field means that the server SHOULD include or exclude these special parts from the FastTransfer stream for top-level messages.

3.2.5.9.2 Upload State

3.2.5.9.2.1 Receiving a RopSynchronizationUploadStateStreamBegin ROP Request

When the client sends the server a **RopSynchronizationUploadStateStreamBegin** ROP (section 2.2.3.2.2.1) request, the server MUST parse the request, as specified in [MS-OXCROPS] section 2.2.13.9.1 and section 2.2.3.2.2.1 of this specification. The server MUST respond with a **RopSynchronizationUploadStateStreamBegin** ROP response, as specified in [MS-OXCROPS] section 2.2.13.9.2 and section 2.2.3.2.2.1 of this specification.

3.2.5.9.2.2 Receiving a RopSynchronizationUploadStateStreamContinue Request

When the client sends the server a **RopSynchronizationUploadStateStreamContinue** ROP (section 2.2.3.2.2.2) request, the server MUST parse the request, as specified in [MS-OXCROPS] section 2.2.13.10.1 and section 2.2.3.2.2.2 of this specification. The server MUST respond with a **RopSynchronizationUploadStateStreamContinue** ROP response, as specified in [MS-OXCROPS] section 2.2.13.10.2 and section 2.2.3.2.2.2 of this specification.

Servers concatenate **StreamData** from all received **RopSynchronizationUploadStateStreamContinue** ROP requests for a given ICS state property.

3.2.5.9.2.3 Receiving a RopSynchronizationUploadStateStreamEnd ROP Request

When the client sends the server a **RopSynchronizationUploadStateStreamEnd** ROP (section 2.2.3.2.2.3) request, the server MUST parse the request, as specified in [MS-OXCROPS] section 2.2.13.11.1 and section 2.2.3.2.2.3 of this specification. The server MUST respond with a **RopSynchronizationUploadStateStreamEnd** ROP response, as specified in [MS-OXCROPS] section 2.2.13.11.2 and section 2.2.3.2.2.3 of this specification.

3.2.5.9.3 Download State

3.2.5.9.3.1 Receiving a RopSynchronizationGetTransferState ROP Request

When the client sends the server a **RopSynchronizationGetTransferState** ROP (section 2.2.3.2.3.1) request, the server MUST parse the request, as specified in [MS-OXCROPS] section 2.2.13.8.1 and section 2.2.3.2.3.1 of this specification. The server MUST respond with a **RopSynchronizationGetTransferState** ROP response, as specified in [MS-OXCROPS] section 2.2.13.8.2 and section 2.2.3.2.3.1 of this specification.

The server MUST ensure that changes to the state of the synchronization context that occur after this ROP do not affect the ICS state that is downloaded through the FastTransfer download context that is returned from this ROP.

The FastTransfer stream in the **RopFastTransferSourceGetBuffer** ROP response buffer associated with the **RopSynchronizationGetTransferState** ROP SHOULD<39> return the initial ICS state for the download context until the end of the FastTransfer stream has been downloaded, or MAY<40> return the checkpoint ICS state that is reflective of the current status. For upload contexts, the FastTransfer stream contains the checkpoint ICS state that is reflective of the current status. After the download is complete, the FastTransfer stream contains the final ICS state.

3.2.5.9.4 Upload

3.2.5.9.4.1 Receiving a RopSynchronizationOpenCollector ROP Request

When the client sends the server a **RopSynchronizationOpenCollector** ROP (section 2.2.3.2.4.1) request, the server MUST parse the request, as specified in [MS-OXCROPS] section 2.2.13.7.1 and section 2.2.3.2.4.1 of this specification. The server MUST respond with a **RopSynchronizationOpenCollector** ROP response, as specified in [MS-OXCROPS] section 2.2.13.7.2 and section 2.2.3.2.4.1 of this specification.

3.2.5.9.4.2 Receiving a RopSynchronizationImportMessageChange ROP Request

When the client sends the server a **RopSynchronizationImportMessageChange** ROP (section 2.2.3.2.4.2) request, the server MUST parse the request, as specified in [MS-OXCROPS] section 2.2.13.2.1 and section 2.2.3.2.4.2 of this specification. The server MUST respond with a **RopSynchronizationImportMessageChange** ROP response, as specified in [MS-OXCROPS] section 2.2.13.2.2 or 2.2.13.2.3, and in section 2.2.3.2.4.2 of this specification.

When the server receives a **RopSynchronizationImportMessageChange** ROP request, the server MUST perform conflict detection on the message, as specified in section 3.1.5.6. The server becomes

responsible for performing conflict resolution on the **RopSaveChangesMessage** ROP ([MS-OXCROPS] section 2.2.6.3), as specified in section 3.1.5.6.2.

The server MUST purge all client-settable properties and subobjects of the Message object prior to returning it in the **OutputServerObject**. Note that any changes to this message made by this ROP or any other ROP that operates on it MUST NOT be persisted until **RopSaveChangesMessage** ROP is called.

ImportFlag Constraints

If the **FailOnConflict** flag of the **ImportFlag** field is set, the server MUST NOT accept conflicting versions of messages.

If the **FailOnConflict** flag of the **ImportFlag** field is not set, the server MUST accept conflicting versions of messages.

Servers SHOULD<41> fail the ROP if unknown flags are set.

3.2.5.9.4.3 Receiving a RopSynchronizationImportHierarchyChange ROP Request

When the client sends the server a **RopSynchronizationImportHierarchyChange** ROP (section 2.2.3.2.4.3) request, the server MUST parse the request, as specified in [MS-OXCROPS] section 2.2.13.4.1 and section 2.2.3.2.4.3 of this specification. The server MUST respond with a **RopSynchronizationImportHierarchyChange** ROP response, as specified in [MS-OXCROPS] section 2.2.13.4.2 or 2.2.13.4.3, and section 2.2.3.2.4.3 of this specification.

Upon successful completion of this ROP, the ICS state on the synchronization context MUST be updated to include a new change number in the **MetaTagCnsetSeen** property (section 2.2.1.1.2).

The server is responsible for conflict detection and resolution, as specified in section 3.1.5.6.

If a conflict is detected, the server MUST resolve it as specified in section 3.1.5.6.2 and return Success. A server can report a conflict using a conflict notification message.

If a conflict has occurred, the server:

- SHOULD NOT update the **MetaTagCnsetSeen** property, and let the clients download a result of conflict resolution.
- MAY generate a conflict notification message. For more details, see section 3.1.5.6.3.
- MUST return a value of **Success** in the **ReturnValue** field.

The server MUST ignore the properties in the **PropertyValues** field, which are also present in the **HierarchyValues** field.

3.2.5.9.4.4 Receiving a RopSynchronizationImportMessageMove ROP Request

When the client sends the server a **RopSynchronizationImportMessageMove** ROP (section 2.2.3.2.4.4) request, the server MUST parse the request, as specified in [MS-OXCROPS] section 2.2.13.6.1 and section 2.2.3.2.4.4 of this specification. The server MUST respond with a **RopSynchronizationImportMessageMove** ROP response, as specified in [MS-OXCROPS] section 2.2.13.6.2 or 2.2.13.6.3, and section 2.2.3.2.4.4 of this specification.

Upon successful completion of this ROP, the ICS state on the synchronization context MUST be updated to include change numbers of messages in the destination folder in either the **MetaTagCnsetSeen** (section 2.2.1.1.2) or **MetaTagCnsetSeenFAI** (section 2.2.1.1.3) property, depending on whether the message is a normal message or an FAI message.

3.2.5.9.4.5 Receiving a RopSynchronizationImportDeletes ROP Request

When the client sends the server a **RopSynchronizationImportDeletes** ROP (section 2.2.3.2.4.5) request, the server MUST parse the request, as specified in [MS-OXCROPS] section 2.2.13.5.1 and section 2.2.3.2.4.5 of this specification. The server MUST respond with a **RopSynchronizationImportDeletes** ROP response, as specified in [MS-OXCROPS] section 2.2.13.5.2 and section 2.2.3.2.4.5 of this specification.

The server MUST ignore requests to delete objects that have already been deleted and SHOULD record deletions of objects that never existed in the server replica, in order to prevent the **RopSynchronizationImportHierarchyChange** (section 2.2.3.2.4.3) or **RopSynchronizationImportMessageChange** (section 2.2.3.2.4.2) ROPs from restoring them back.

To minimize the possibility of putting replicas into a desynchronized state and because the protocol does not notify clients as to what part of an operation has succeeded, servers are responsible for making a reasonable prediction as to whether all deletions will succeed. And, if a deletion will not succeed, the server SHOULD fail the ROP before performing any deletions, as opposed to partially completing the ROP.

Servers SHOULD fail the ROP if unknown **ImportDeleteFlags** flag bits are set.

3.2.5.9.4.6 Receiving a RopSynchronizationImportReadStateChanges ROP Request

When the client sends the server a **RopSynchronizationImportReadStateChanges** ROP (section 2.2.3.2.4.6) request, the server MUST parse the request, as specified in [MS-OXCROPS] section 2.2.13.3.1 and section 2.2.3.2.4.6 of this specification. The server MUST respond with a **RopSynchronizationImportReadStateChanges** response, as specified in [MS-OXCROPS] section 2.2.13.3.2 and section 2.2.3.2.4.6 of this specification.

The **RopSynchronizationImportReadStateChanges** ROP is a batch variant of the **RopSetMessageReadFlag** ROP ([MS-OXCROPS] section 2.2.6.11), which updates the ICS state as well. The result of changing the read state message by message by using the **RopSetMessageReadFlag** ROP MUST be identical to changing the read state in bulk by using the **RopSynchronizationImportReadStateChanges** ROP.

Requests to change the read state of FAI messages MUST be ignored. Upon successful completion of this ROP, the ICS state on the synchronization context MUST be updated by adding the new change number to the **MetaTagCnsetRead** property (section 2.2.1.1.4).

To minimize the possibility of putting replicas into a desynchronized state and because the protocol does not notify clients as to what part of an operation has succeeded, servers are responsible for making a reasonable prediction as to whether all read state changes will succeed. And, if a read state change will not succeed, the server SHOULD fail the ROP before performing any read state changes, as opposed to partially completing the ROP.

3.2.5.9.4.7 Receiving a RopGetLocalReplicaIds ROP Request

When the client sends the server a **RopGetLocalReplicaIds** ROP (section 2.2.3.2.4.7) request, the server MUST parse the request, as specified in [MS-OXCROPS] section 2.2.13.13.1 and section 2.2.3.2.4.7 of this specification. The server MUST respond with a **RopGetLocalReplicaIds** ROP response, as specified in [MS-OXCROPS] section 2.2.13.13.2 or 2.2.13.13.3, and section 2.2.3.2.4.7 of this specification.

A server can limit the number of IDs that can be allocated in one batch to prevent malicious clients from reserving too many IDs with the intent of causing a denial-of-service attack by depleting the set of available IDs. A server can limit the maximum number of IDs that can be allocated in one batch to the upper limit of the range recommended to clients, as specified in section 3.3.5.8.12.

3.2.5.9.4.8 Receiving a RopSetLocalReplicaMidsetDeleted ROP Request

When the client sends the server a **RopSetLocalReplicaMidsetDeleted** ROP (section 2.2.3.2.4.8) request, the server MUST parse the request, as specified in [MS-OXCROPS] section 2.2.13.12.1 and section 2.2.3.2.4.8 of this specification. The server MUST respond with a **RopSetLocalReplicaMidsetDeleted** ROP response, as specified in [MS-OXCROPS] section 2.2.13.12.2 and section 2.2.3.2.4.8 of this specification.

A server MUST add ranges of IDs supplied through this ROP to the deleted item list. By adding ranges of IDs to the deleted item list, the server is able to compress the deleted item list by using the **IDSET** structure optimization algorithm specified in section 3.1.5.5.

3.2.5.10 Effect of Property and Subobject Filters on Download

Property and subobject filters specified during the configuration of a download operation only have an effect on the objects that are directly included in the scope of the operation. For example:

- Specifying a property in the **PropertyTags** field of the request buffer of a **RopFastTransferSourceCopyProperties** ROP (section 2.2.3.1.1.2)whose **InputServerObject** field contains an Attachment object affects the set of properties copied for this attachment, but not its embedded message or any attachments that it might contain.
- Specifying the **PidTagFolderAssociatedContents** property ([MS-OXPROPS] section 2.690) in the **PropertyTags** field of the request buffer of a **RopFastTransferSourceCopyTo** ROP (section 2.2.3.1.1.1)whose **InputServerObject** field contains a Folder object only excludes FAI Message objects from copying this specific folder, but not any of its descendant folders.
- Specifying the **PidTagMessageRecipients** property ([MS-OXPROPS] section 2.786) in the **PropertyTags** fields of the request buffer of a **RopSynchronizationConfigure** ROP (section 2.2.3.2.1.1) excludes **recipient** subobjects from all message changes downloaded in that operation, but it does not affect recipients of embedded messages that their attachments might have.

Regardless of property filters specified at operation configuration time, certain properties MUST always be excluded from output. For details about the properties to exclude from output, see section 3.2.5.12.

At the same time, directives to include or exclude properties and subobjects supplied through flags do have an effect on downloaded objects at all levels. For example:

- Specifying the **CopySubfolders** flag of the **CopyFlag** field, as specified in section 2.2.3.1.1.4.1, includes all subfolders of the current folder into the operation scope.
- Specifying the **SendEntryId** flag of the **CopyFlag** field includes all identification properties for all objects being downloaded.

Whenever subobject filters have an effect, servers MUST output a **MetaTagFXDelProp** meta-property (section 2.2.4.1.5.1) immediately before outputting subobjects of a particular type, to differentiate between the cases where a set of subobjects (such as attachments or recipients) was filtered in, but was empty, and where it was filtered out. For example:

- Specifying the **PidTagMessageRecipients** meta-property ([MS-OXPROPS] section 2.786) in the **PropertyTags** field of the request buffer of the **RopFastTransferSourceCopyProperties** ROP (section 2.2.3.1.1.2) where the **InputServerObject** field contains a Message object, directs the server to output the **MetaTagFXDelProp** (section 2.2.4.1.5.1) and **PidTagMessageRecipients** properties before outputting recipients of that message, even if there are no recipients.

The protocol does not support incremental download of subobjects. Subobjects of a particular type are either filtered out, in which case the **MetaTagFXDelProp** meta-property MUST NOT be output, or are filtered in; that is, they MUST be output one after another, prefixed by the **MetaTagFXDelProp** meta-property.

3.2.5.11 Properties to Ignore on Upload

Unless specified otherwise in property list restriction tables, properties that belong to the **provider-defined internal nontransmittable** range, as specified in [MS-OXPROPS] section 1.3.3, MUST be ignored on upload.

3.2.5.12 Properties to Ignore on Download

Unless specified otherwise in property list restriction tables, **propValue** elements of FastTransfer streams, as specified in section 2.2.4.3.21, that belong to the provider-defined internal nontransmittable range, as specified in [MS-OXPROPS] section 1.3.3, MUST be excluded from download.

3.2.6 Timer Events

None.

3.2.7 Other Local Events

None.

3.3 Client Details

This section provides client-specific details related to bulk data transfer. When participating in synchronization, the client has several responsibilities in addition to the actual act of synchronization. These include: ID assignment, change tracking, conflict resolution, and ICS state storage.

3.3.1 Abstract Data Model

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

This protocol includes the following ADM types and elements:

- **Global.Handle**, as specified in [MS-OXCRPC] section 3.1.1.
- **Session context cookie**<42>, as specified in [MS-OXCMAPIHTTP] section 3.1.1.
- **MessagingObject**, as specified in section 3.1.1.2. Additional elements for the **MessagingObject** ADM type are defined in section 3.3.1.1.

3.3.1.1 Per Messaging Object

Messaging objects are represented by the **MessagingObject** ADM type. The following abstract object elements are maintained by the client for each **MessagingObject** ADM type:

Client.MessagingObject.ForeignIdentifier: An **XID** structure, as specified in section 2.2.2.2, that identifies changes to objects in the local replica. For more details about foreign identifiers, see section 3.3.5.2.3.

3.3.2 Timers

None.

3.3.3 Initialization

None.

3.3.4 Higher-Layer Triggered Events

3.3.4.1 Downloading Messaging Objects Using FastTransfer

Clients can efficiently download copies of specified folders, messages or attachments using the binary format known as a FastTransfer stream. The following steps MUST be taken by a client to download copies of these messaging objects from the server using FastTransfer ROPs and FastTransfer streams:

1. Obtain a handle to a messaging object whose contents are requested, or obtain a handle to a messaging object that the client will download a copy of. To obtain the handle of a new messaging object, use the **OutputHandleIndex** field from the **RopCreateMessage** ROP ([MS-OXCROPS] section 2.2.6.2) response buffer. For more details about obtaining a handle to an existing messaging object, see [MS-OXCMSG] section 3.1.4.1.

2. Send the **RopFastTransferSourceCopy*** ROP request to create the FastTransfer download context on the server and define the parameters and the scope of the operation. The **RopFastTransferSourceCopy*** ROPs are specified in section 2.2.3.1.1.

3. Optionally, send a **RopTellVersion** ROP (section 2.2.3.1.1.6) request, if performing a server-to-client-to-server upload, as specified in section 3.3.4.2.1. Additional details about sending a **RopTellVersion** ROP request are specified in section 3.3.5.7.2.

4. Iteratively send **RopFastTransferSourceGetBuffer** ROP (section 2.2.3.1.1.5) requests on the FastTransfer context to retrieve the FastTransfer stream with serialized messaging objects. Additional details about sending a **RopFastTransferSourceGetBuffer** ROP request are specified in section 3.3.5.7.1.

5. Send a **RopRelease** ROP ([MS-OXCROPS] section 2.2.15.3) request to release the messaging object and FastTransfer context obtained in steps 1 and 2.

3.3.4.2 Uploading Messaging Objects Using FastTransfer

Clients not only can download copies of specified folders, messages or attachments using FastTransfer ROPs and FastTransfer streams, clients can also upload copies of these messaging objects as well. The following steps MUST be taken by a client to upload copies of messaging objects to the server using **FastTransfer** ROPs and FastTransfer streams:

1. Obtain a handle to a messaging object, for which appending or replacing properties and/or subobjects is requested. To obtain a handle to a new messaging object, use the **OutputHandleIndex** field from the **RopCreateMessage** ROP ([MS-OXCROPS] section 2.2.6.2) response buffer. For more details about obtaining a handle to an existing messaging object, see [MS-OXCMSG] section 3.1.4.1.

2. Send the **RopFastTransferDestinationConfigure** ROP (section 2.2.3.1.2.1) to create a FastTransfer upload context on the server and define the parameters of the operation.

3. Optionally, send the **RopTellVersion** ROP (section 2.2.3.1.1.6) if performing a server-to-client-to-server upload, as specified in section 3.3.4.2.1. Additional details about sending a **RopTellVersion** ROP request are specified in section 3.3.5.7.2.

4. Iteratively send the **RopFastTransferDestinationPutBuffer** ROP (section 2.2.3.1.2.2) on the FastTransfer context to upload the FastTransfer stream with the serialized messaging objects.

5. Send the **RopRelease** ROP ([MS-OXCROPS] section 2.2.15.3) to release the messaging object and the FastTransfer context obtained in steps 1 and 2.

In step 4, if a client simply resends the stream that it is getting through the FastTransfer download, it can consider using an optimized server-to-client-to-server upload process, as specified in section 3.3.4.2.1.

3.3.4.2.1 Server-to-Client-to-Server Upload

To optimize copying messaging objects between two different mailboxes on two different servers by using FastTransfer upload paired with FastTransfer download, a client can specify the **ForUpload** flag in the **SendOptions** field of the **RopFastTransferSourceCopy*** ROPs, as specified in section 2.2.3.1.1, which instructs the source server to produce a FastTransfer stream that is optimized for the destination server. By setting the **ForUpload** flag, the client instructs the server to transmit all string properties in an untranslated format to preserve full fidelity on the destination server.

Clients MUST NOT parse the FastTransfer stream produced by the source server, as it can contain optimizations and not adhere to the grammar specified in section 2.2.4.

Clients MUST use the following steps to execute server-to-client-to-server copying:

1. Send one of the **RopFastTransferSourceCopy*** ROP requests to server A to configure a FastTransfer download context, while setting the **ForUpload** flag in the **SendOptions** field.
2. Send the **RopFastTransferDestinationConfigure** ROP (section 2.2.3.1.2.1) request to server B to configure a FastTransfer upload context.
3. Send the **RopTellVersion** ROP (section 2.2.3.1.1.6) request on the FastTransfer download context with a version of server B.
4. Send the **RopTellVersion** ROP request on the FastTransfer upload context with a version of server A.
5. Iteratively send the **RopFastTransferSourceGetBuffer** ROP (section 2.2.3.1.1.5) requests on the FastTransfer download context, followed by the **RopFastTransferDestinationPutBuffer** ROP (section 2.2.3.1.2.2) requests on the FastTransfer upload context, until there is no more data.
6. Release both FastTransfer contexts.

3.3.4.3 Synchronizing Incremental Changes

ICS is used to determine differences between two folder hierarchies or two sets of content, and can upload or download information about the differences in a single session. The following figure shows the high-level steps involved in ICS.

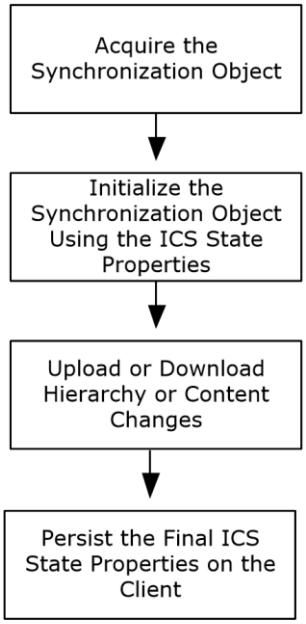


Figure 1: Steps in Incremental Change Synchronization

1. The client acquires the synchronization object by using the **RopSynchronizationConfigure** ROP (section 2.2.3.2.1.1) for download operations, or the **RopSynchronizationOpenCollector** ROP (section 2.2.3.2.4.1) for upload operations.

2. The client initializes the synchronization object by uploading the ICS state properties, as specified in section 3.3.4.3.1.

3. To upload content changes to the server, the client sends data using the steps specified in section 3.3.4.3.3.2.2. To upload hierarchy changes to the server, the client sends data using the steps specified in section 3.3.4.3.3.1. To download content or hierarchy changes from the server, the client uses the steps specified in section 3.3.4.3.2.

4. After the content synchronization is complete, the client receives the final ICS state properties, which it persists locally.

3.3.4.3.1 Uploading the ICS State

The ICS state properties, as specified in section 2.2.1.1, are used to determine the differences between the messaging objects on the client and the server. By using the ICS state properties, only differences that are relevant to a client are downloaded and the same information is only downloaded once.

The client passes the ICS state properties to the server immediately after configuring a synchronization context for download or upload by sending the **RopSynchronizationConfigure** ROP (section 2.2.3.2.1.1) or the **RopSynchronizationOpenCollector** ROP (section 2.2.3.2.4.1), respectively.

After the synchronization context is acquired, the client MUST send the initial ICS state properties, as specified in section 2.2.1.1, to the server before executing any other ROPs on the synchronization context.

During the first synchronization of a synchronization scope, the client MUST send the initial ICS state properties as zero-length byte arrays.

The following table summarizes the requirements for including the ICS state properties depending on the synchronization context of the operation.

ICS state property	Hierarchy download	Contents download	Hierarchy upload	Contents upload
MetaTagIdsetGiven (section 2.2.1.1.1)	MUST	MUST	Not applicable	Not applicable
MetaTagCnsetSeen (section 2.2.1.1.2)	MUST	MUST	SHOULD	SHOULD
MetaTagCnsetSeenFAI (section 2.2.1.1.3)	Not applicable	MUST	Not applicable	SHOULD
MetaTagCnsetRead (section 2.2.1.1.4)	Not applicable	MUST	Not applicable	SHOULD

Uploading the ICS state properties is done sequentially, property by property. The order in which properties are uploaded does not matter. The upload of each property MUST be initiated by sending the **RopSynchronizationUploadStateStreamBegin** ROP (section 2.2.3.2.2.1) request, followed by one or more **RopSynchronizationUploadStateStreamContinue** ROP (section 2.2.3.2.2.2) requests. The upload is finished with the **RopSynchronizationUploadStateStreamEnd** ROP (section 2.2.3.2.2.3).

At the end of the synchronization operation, the client receives a new ICS state from the server, commonly referred to as the final ICS state. Updated ICS state properties can also be returned to the client in ROP responses, or through the checkpointing process, as specified in section 3.3.5.6. It is the responsibility of the client to persist the ICS state properties.

3.3.4.3.2 Downloading Changes Using ICS

The following steps MUST be taken by a client when downloading mailbox differences from a server using ICS:

1. Obtain a handle to a Folder object, for which synchronization is being requested. For details about obtaining a folder handle, see [MS-OXCFOLD].
2. Send the **RopSynchronizationConfigure** ROP (section 2.2.3.2.1.1) request to create a synchronization download context on the server and define the parameters and the scope of the operation.
3. Send the **RopSynchronizationUploadStateStreamBegin** (section 2.2.3.2.2.1), **RopSynchronizationUploadStateStreamContinue** (section 2.2.3.2.2.2), and **RopSynchronizationUploadStateStreamEnd** ROP (section 2.2.3.2.2.3) requests to upload the initial ICS state information to the synchronization context.
4. Iteratively send the **RopFastTransferSourceGetBuffer** ROP (section 2.2.3.1.1.5) request on the synchronization download context to retrieve the FastTransfer stream of the mailbox differences and the final ICS state.
5. Persist the ICS state.
6. Send the **RopRelease** ROP ([MS-OXCROPS] section 2.2.15.3) request to release the Folder object and the synchronization download context obtained in steps 1 and 2.

3.3.4.3.3 Uploading Changes Using ICS

The client uploads the initial ICS state and downloads the final/checkpoint ICS state when performing a synchronization upload. Clients can perform a synchronization upload without uploading the initial ICS state properties into a synchronization upload context, because the behavior of the **RopSynchronizationImport*** ROPs do not depend on the initial ICS state. In that case, a server can download the changes uploaded in this session during the subsequent ICS downloads.

The following figure shows the primary processes taking place during an upload operation. The sections that follow describe the details within the Send Data process.

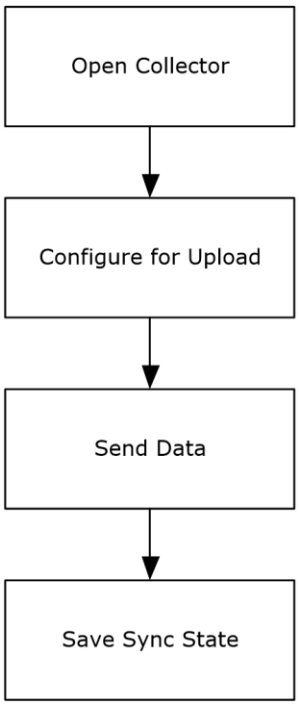


Figure 2: Upload operation

The following steps elaborate on the steps in the figure and MUST be taken by a client when uploading mailbox differences to a server:

1. Obtain a handle to the Folder object, as specified in [MS-OXCFOLD], that will be synchronized.
2. Send a **RopSynchronizationOpenCollector** ROP request (section 2.2.3.2.4.1) to create a synchronization upload context on the server and to define parameters and the scope of an operation.
3. The client SHOULD send the **RopSynchronizationUploadStateStreamBegin** ROP (section 2.2.3.2.2.1), the **RopSynchronizationUploadStateStreamContinue** ROP (section 2.2.3.2.2.2), and the **RopSynchronizationUploadStateStreamEnd** ROP (section 2.2.3.2.2.3) request to upload the initial ICS state information to the synchronization context.
4. Upload changes, moves, and deletes of individual objects within the synchronized Folder object through **RopSynchronizationImport*** ROPs, while passing the synchronization upload context obtained in step 2. Uploading hierarchy changes is specified in section 3.3.4.3.3.1. Uploading content changes is specified in section 3.3.4.3.3.2.
5. The client SHOULD obtain the final ICS state by doing the following:

- Acquire a separate FastTransfer download context for a checkpoint ICS state by using the **RopSynchronizationGetTransferState** ROP (section 2.2.3.2.3.1) and passing the synchronization upload context obtained in step 2 in the request buffer.
- Perform FastTransfer download step 4, as specified in section 2.2.3.1.1, on the FastTransfer download context acquired in the first bullet point.
- Release the FastTransfer download context obtained in the first bullet point.

6. Persist the ICS state.
7. Send the **RopRelease** ROP ([MS-OXCROPS] section 2.2.15.3) request to release the Folder object and the synchronization upload context obtained in steps 1 and 2.

The client can elect not to upload/download the ICS states in steps 3 and 5. For details on how that would affect responsibilities of the protocol roles, see section 3.3.4.3.3.

The following table lists the common return values from the **RopSynchronizationImport*** ROPs that clients SHOULD have special processing for.

Value	Description
Success	No error occurred, or a conflict has been resolved**.**
NoParentFolder	An attempt is being made to upload a hierarchy change for a folder whose parent folder does not yet exist.
ObjectDeleted	An attempt is being made to import a message change to a message that has been deleted. The client SHOULD subsequently ignore the failure because it is just a warning.
IgnoreFailure	An attempt is being made to import a change that the server already has. For example, if the client calls the **RopSynchronizationImportMessageChange** ROP, as specified in section 2.2.3.2.4.2, and uploads a change and then issues the exact same **RopSynchronizationImportMessageChange ROP request** to the server with the exact same message with the exact same change, the server returns this value because it either already has the change or has a version of the message that supersedes that change, so it does not need the change.

The complete list of error codes is specified in [MS-OXCDATA] section 2.4.

3.3.4.3.3.1 Hierarchy Upload

The following sections specify best practices for uploading hierarchy modifications and deletions that were tracked by the client while the client was **offline**.

The Send Data process (as shown in Figure 4) is illustrated in Figure 5.

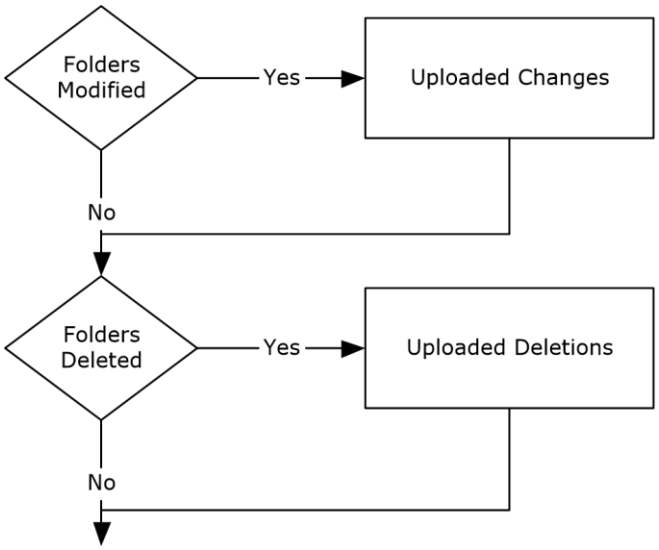


Figure 3: Send Data process

When uploading hierarchy differences, the client sends the following ROP requests:

- **RopSynchronizationImportHierarchyChange** ROP (section 2.2.3.2.4.3)
- **RopSynchronizationImportDeletes** ROP (section 2.2.3.2.4.5)

3.3.4.3.3.1.1 Uploading Hierarchy Changes

New and modified folders are uploaded in the same manner. A client MUST collect all properties that are stored on the local replica, and use the synchronization upload context and the **RopSynchronizationImportHierarchyChange** ROP (section 2.2.3.2.4.3), to transmit this information to the server. When public folders are uploaded, the synchronization upload context is opened by using the folder that is being synchronized. When this happens, a **PidTagParentSourceKey** property (section 2.2.1.2.6) with a zero-length value is used to denote that the folder properties belong to the folder from which the synchronization upload context was opened. A move of a folder from one parent to another is modeled as a modification of a folder, where the value of the **PidTagParentSourceKey** property of the folder changes to reflect the new parent.

There is no mechanism to represent conflicts on hierarchy. As such, the server MUST apply "last writer wins" semantics to hierarchy change uploads. The last writer wins algorithm is specified in section 3.1.5.6.2.2.

Clients SHOULD ignore the following errors, which indicate that the server did not apply the changes:

Value	Description
ObjectDeleted	An object or its parent folder has already been deleted.
IgnoreFailure	The change was ignored, as it has been superseded by another change.

The complete list of error codes is specified in [MS-OXCDATA] section 2.4.

3.3.4.3.3.1.2 Uploading Hierarchy Deletions

Folder deletions are performed by transmitting the **PidTagSourceKey** property (section 2.2.1.2.5) for the folder to be removed by using the synchronization upload context and the **RopSynchronizationImportDeletes** ROP (section 3.3.5.8.10).

The server MUST stop processing deletion operations upon encountering the first error; therefore, a client MUST be prepared to retry the operations on a newly initialized synchronization upload context if a failure is encountered.

Errors can occur during bulk operations that require additional effort to disambiguate and rectify. These errors can occur more often when the user is not the owner of the mailbox.

3.3.4.3.3.2 Content Upload

The following sections specify best practices for uploading content modifications, read/unread state changes, deletions, and move operations that were performed while the client was offline. The following figure shows this process.

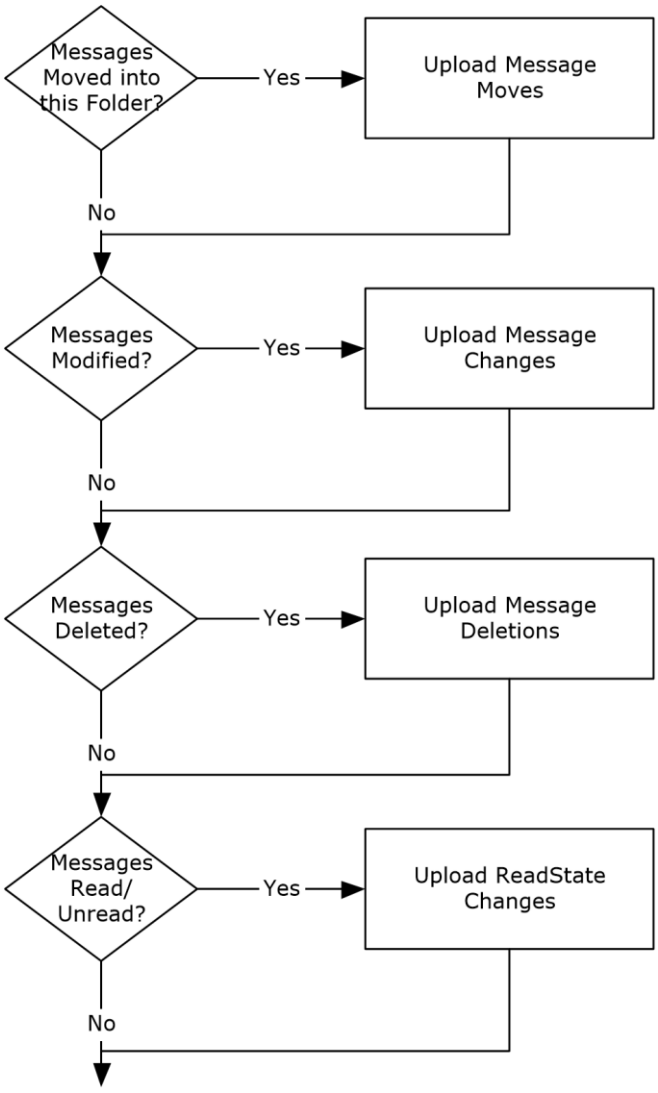


Figure 4: Best practices for Message Changes

When uploading content differences, the client can send any combination of the following ROP requests:

- **RopSynchronizationImportMessageChange** (section 2.2.3.2.4.2). Imports new messages or changes to existing messages.
- **RopSynchronizationImportMessageMove** (section 2.2.3.2.4.4). Communicates the movement of messages between folders within the same mailbox.
- **RopSynchronizationImportDeletes** (section 2.2.3.2.4.5). Imports deletions of messages.
- **RopSynchronizationImportReadStateChanges** (section 2.2.3.2.4.6). Imports changes to the read state of messages.

These ROPs do not have to be sent in any specific order and can be mixed together. For example, all the deletions do not have to be uploaded before all the message moves, and all the message changes do not have to be uploaded before all the deletions.

3.3.4.3.3.2.1 Uploading Moves

3.3.4.3.3.2.1.1 Moves and Modifications

Message moves MUST be performed using a synchronization upload context of the folder to which the message was moved (the destination folder). To synchronize a message move, use the **RopSynchronizationImportMessageMove** ROP (section 2.2.3.2.4.4). The client specifies the source folder information.

3.3.4.3.3.2.1.2 Avoiding Duplicate Uploads

When a client that is using a local replica sends a message by using a server with a major version of less than eight (8), a new message is created in a server folder, the contents of the local message are copied into the server message, and the message is submitted (on the first upload). After the submission is complete, the item is moved into the **Sent Items folder** in the local replica. Sometime later, the Sent Items folder is synchronized. The item is then uploaded as it is a new item in the replica (on the second upload) due to the fact that the folder to which the message was originally submitted is not within the user's mailbox (and therefore not part of the replica).

Servers with a major version of 8 and above implement the following alternative method to diminish the impact of the second upload. For details about determining server version, see section 1.7. Because the client assigns server IDs to all items it creates in the local replica, the message in the **Outbox folder** has a valid server ID. When the client is uploading the message for sending, it adds an additional property to the message, **PidTagTargetEntryId** ([MS-OXOMSG] section 2.2.1.76). When a version 8 (or later) server observes this property, it places a mirror copy of the message in the user's server Outbox and gives it the ID specified in the **PidTagTargetEntryId** property. In this way, when the client synchronizes the folder holding sent items, it can be synchronized as a move (from the Outbox folder to the folder holding sent items) as opposed to a new item (in the folder holding sent items).

3.3.4.3.3.2.2 Uploading Modifications

The following sections specify the processes for uploading message modifications. Conflict detection is defined in section 3.1.5.6.1.

When a message has been moved and modified in the offline message store, a client MUST apply modifications after moving the message, as specified in section 3.3.4.3.3.2.1.1.

3.3.4.3.3.2.2.1 Full Item Upload

Message changes are uploaded by using the **RopSynchronizationImportMessageChange** ROP (section 2.2.3.2.4.2), followed by copying all properties to the message by using the **RopSetProperties** ROP ([MS-OXCROPS] section 2.2.8.6), as specified in [MS-OXCPRPT] section 2.2.5, and persisting the changes using the **RopSaveChangesMessage** ROP, as specified in [MS-OXCMSG] section 2.2.3.3.

Clients SHOULD ignore the following errors (returned from the **RopSynchronizationImportMessageChange** ROP and the **RopSaveChangesMessage** ROP, or both), which indicate that the server did not apply the changes.

Value	Description
ObjectDeleted	An object or its parent folder has already been deleted.
IgnoreFailure	The change was ignored, as it has been superseded by another change.

The complete list of error codes is specified in [MS-OXCDATA] section 2.3.

A client SHOULD have a strategy to deal with non-transient errors to prevent them from occurring in subsequent synchronization attempts. That strategy can be to move items that fail to upload to a local folder that is not synchronized. In addition, for non-new items, the client can attempt to bring down the previous version of the item in order to get a good version of the item back in the user's offline message store.

When detecting conflicting changes, a server MUST perform conflict resolution as defined in section 2.2.1.4.1.

3.3.4.3.3.2.2.2 Partial Item Upload

To improve wire efficiency, a client SHOULD track offline changes in such a way that these can be uploaded individually without having to transmit the full item. One strategy is for a client to apply the changes by using standard Message object and Attachment object protocol calls, as specified in [MS-OXCMSG], such as the **RopGetPropertiesSpecific** ROP ([MS-OXCROPS] section 2.2.8.3), the **RopSetProperties** ROP ([MS-OXCROPS] section 2.2.8.6), the **RopSetReadFlags** ROP ([MS-OXCROPS] section 2.2.6.10), and the **RopDeleteProperties** ROP ([MS-OXCROPS] section 2.2.8.8), to apply changes directly to the server replica. Then, the client uses the **RopSaveChangesMessage** ROP ([MS-OXCROPS] section 2.2.6.3) together with the **RopGetPropertiesSpecific** ROP ([MS-OXCROPS] section 2.2.8.3) to get the new PCL and change number values for the message, as specified in sections 2.2.2.3 and 2.2.2.1, respectively. The client SHOULD then modify the message PCL, message change number, and ICS state on the client in a way that would prevent the item from being downloaded, as specified in section 3.1.5.6.

3.3.4.3.3.2.3 Uploading Deletes

Message deletions are performed by transmitting the **PidTagSourceKey** property (section 2.2.1.2.5) for the messages to be removed by using the **RopSynchronizationImportDeletes** ROP (section 2.2.3.2.4.5) on a synchronization upload context of the folder that contains those messages.

A client SHOULD batch multiple **PidTagSourceKey** property entries in the **PropertyValues** field of the **RopSynchronizationImportDeletes** ROP when deleting multiple messages to improve wire efficiency.

The server MUST stop processing deletion operations upon encountering the first NotFound error, as specified in [MS-OXCDATA] section 2.4, so a client MUST be prepared to retry the operations on a newly initialized synchronization upload context if a failure is encountered.

Errors can occur during bulk operations that require additional effort to disambiguate and rectify. These errors occur more often when the user is not the owner of the mailbox.

3.3.4.3.3.2.4 Uploading Read/Unread State Changes

Message read/unread state is uploaded by transmitting one **MessageReadState** structure ([MS-OXCROPS] section 2.2.13.3.1.1) per message, which changes the status by using the synchronization upload context and the **RopSynchronizationImportReadStateChanges** ROP (section 2.2.3.2.4.6).

For each message that has a read state change, the client MUST specify the **MessageId** field that represents the **PidTagSourceKey** property (section 2.2.1.2.5) for the message, as well as the **MarkAsRead** field, indicating the updated read state, as specified in section 2.2.3.2.4.6.1.

Errors can occur during bulk operations that require additional effort to disambiguate and rectify. These errors occur more often when the user is not the owner of the mailbox.

3.3.4.3.4 Downloading the ICS State

The client can download the ICS state of the server using the following methods:

- Sending the **RopSynchronizationGetTransferState** ROP (section 2.2.3.2.3.1) to retrieve the current ICS state of the server.
- Parsing the final ICS state of the server from the **state** element of the FastTransfer stream, as specified in section 2.2.4.3.25.
- Using client side checkpointing, as specified in section 3.3.5.6.

3.3.5 Message Processing Events and Sequencing Rules

3.3.5.1 Order of Operations

When performing synchronization, whether to perform upload operations before download operations is implementation-dependent and external to the Bulk Data Transfer Protocol.<43> However, performing upload before download allows conflicts to be resolved before data is received by the client. In this way, wire efficiency can be increased, as an additional post-resolution upload is not required.

3.3.5.2 Creating Objects and Identifying Changes on the Local Replica

The following three alternative mechanisms are available to clients that require the ability to create objects in their local replica without having immediate contact with the server to upload the differences. This is also known as working offline.

3.3.5.2.1 Client-Assigned Internal Identifiers

When using this most preferred approach, clients MUST send a request to a server to allocate a range of internal identifiers for their exclusive use by using **RopGetLocalReplicaIds** ROP. Once the range is allocated, a client can stay offline and use identifiers from that range until the range is exhausted, at which point the client would have to allocate a new range by connecting to the server and executing the **RopGetLocalReplicaIds** ROP before being able to assign new client-assigned internal identifiers. Clients can then assign these IDs to any new folders or messages within their local replica and communicate these assignments back when performing ICS upload by using the **RopSynchronizationImportHierarchyChange** ROP (section 2.2.3.2.4.3) or the **RopSynchronizationImportMessageChange** ROP (section 2.2.3.2.4.2). Note that these IDs MUST NOT be used for change numbers as it would result in change numbers that did not increase per namespace replica, or clients having different ranges from the same namespace, leading to undefined server behavior.

Clients MUST generate foreign identifiers to identify changes to objects in the local replica, as specified in section 3.3.5.2.3.

This mechanism is being serviced by two ROPs, **RopGetLocalReplicaIds** (section 2.2.3.2.4.7) and **RopSetLocalReplicaMidsetDeleted** (section 2.2.3.2.4.8).

To help compression of **IDSET** structures and to alleviate fragmentation of the deleted item list, if a server maintains an **IDSET** for a folder, clients SHOULD assign consecutive IDs from the allocated range to messages within the same folder. This can be achieved by allocating a contiguous subset of allocated IDs to each folder.

Clients MUST report IDs assigned to objects in a client replica that were deleted without ever being uploaded through the **RopSynchronizationImportDeletes** ROP.

Clients MUST report ranges of server-allocated IDs, which will never be used for any messages in a folder, through the **RopSetLocalReplicaMidsetDeleted** ROP. For more details, see section 3.3.5.8.13.

3.3.5.2.2 Use Online Mode ROPs

In this approach, clients upload objects created in their local replica by using the regular, non-synchronization ROPs, such as **RopCreateFolder** or **RopCreateMessage**, as specified in [MS-OXCROPS], which makes servers assign internal identifiers as usual. The following are the limitations of this mode:

- Clients do not have server-accepted identifiers for objects until after they are uploaded to a server.
- Clients do not control internal identifiers assigned to objects and changes by a server.
- Clients cannot set values of special properties, such as the **PidTagLastModificationTime** property ([MS-OXPROPS] section 2.755).
- Clients are entirely responsible for updating the ICS state to prevent uploaded objects from being downloaded during a subsequent synchronization download operation.

3.3.5.2.3 Foreign Identifiers

Clients MUST generate foreign identifiers to identify changes to objects in the local replica. Foreign identifiers are represented as **XID** structures, as specified in section 2.2.2.2, and MUST NOT have the same byte length as **GID** structures ([MS-OXCDATA] section 2.2.1.3); that is, the number of bytes in the **LocalId** field that follows a **NamespaceGuid** field in the **XID** structure MUST be different from the size of **GLOBCNT** structure, as specified in section 2.2.2.5, which is 6 bytes. At the same time, foreign identifiers that share values for the **NamespaceGuid** field MUST have **LocalId** fields of the same length.

Clients MUST create foreign identifiers within the values of the **NamespaceGuid** fields they generated, and MUST NOT use any **REPLGUID** structures returned by a server for that purpose.

Foreign identifiers MUST have the same qualities as internal identifiers: they MUST be unique, MUST NOT ever be reused and MUST be guaranteed to increase for any new change, or use a different GUID. This is important for conflict detection, as specified in section 3.1.5.6.1.

3.3.5.3 Back-in-Time Detection

Due to the internal identifier allocation specified in section 3.3.5.2.1 and section 3.3.5.2.2 and the fact that a client MUST NOT reuse internal identifiers, the client MUST implement some mechanism to detect server rollback. One possible source of rollback is when a mailbox database is restored from a backup to a previous state.

A sample implementation is a property on both the local replica and the server that stores a counter specific to each replica (2). When the replica (2) connects to the server, it verifies that the counter is greater than or equal to the server counter. If the client counter is ever less than the server counter, rollback has occurred and that client replica (2) is abandoned.

3.3.5.4 Mailbox Validation

A client MUST NOT let a replica (2) of one mailbox synchronize with a different mailbox. For this reason, a client MUST identify the mailbox associated with any given replica (2). This can be accomplished by using the mailbox instance GUID, which is returned by the **RopLogon** ROP ([MS-OXCROPS] section 2.2.3.1). The mailbox is associated with the local replica and compared to the mailbox GUID after the **RopLogon** ROP completed.

Mailbox validation is particularly important in a disaster recovery scenario. For example, assume a database is lost and an interim mailbox is created while database recovery is in progress. In this scenario, the client receives a new mailbox instance GUID in the **RopLogon ROP response** while connecting to the interim mailbox. In a possible implementation, this causes the client to switch to online mode to keep the replica (2) from synchronizing with more than one mailbox. Once the database recovery is complete, the original mailbox instance GUID is returned in the **RopLogon** ROP response, and the client can switch back to cached mode.

3.3.5.5 Determining the Synchronization Scope

To be able to perform an ICS download of mailbox data, a client MUST subdivide all necessary synchronization work into smaller pieces, which clearly define boundaries of synchronization operations in the terms supported by the ICS protocol (see the **RopSynchronizationConfigure** ROP, as specified in section 2.2.3.2.1.1). Synchronization scope is determined by using the following variables:

- **Synchronization type** (**Hierarchy** or **Contents**), as indicated by the **SynchronizationType** enumeration of the **RopSynchronizationConfigure** ROP request buffer.
- Folder within the mailbox, as indicated by the **InputServerObject** field of the **RopSynchronizationConfigure** ROP request.
- Restrictions on messages within the folder that are included in the scope (for content synchronization operations only), as indicated by the **RestrictionData** field of the **RopSynchronizationConfigure** ROP request.

Synchronization for each of the scopes can be performed independently. For each synchronization scope, a client MUST persist the corresponding ICS state and pass it along when configuring a synchronization operation, as specified in section 2.2.3. ICS state does not reflect the synchronization scope it belongs to. Therefore, a client MUST ensure that the ICS state it passes to a server corresponds to the synchronization scope that it was originally obtained for.

Examples of synchronization scopes include the following:

- Folder hierarchy that starts with folder X
- All contents of folder Z
- All unread messages in folder Y that were received within the last three days

Note that the set of messaging objects that are considered for ICS operation can be further limited with flags, such as **Normal** or **FAI** set in the **SynchronizationFlags** field of the **RopSynchronizationConfigure** ROP. However, these flags do not modify the synchronization scope; they just filter the output produced by an operation.

For example, consider the following ICS operation:

- IcsDownload(icsStateX, Normal | FAI) => (diffNormal diffFAI, icsStateZ)

This operation outputs differences for all the messages in a folder. Compare it with the following sequence of ICS operations:

1. IcsDownload(icsStateX, Normal) => (diffNormal, icsStateY)

2. IcsDownload(icsStateY, FAI) => (diffFAI, icsStateZ)

This sequence is correct and it produces the same result as the previous single step operation.

The following sequence, however, is incorrect, because it uses a different synchronization scope (by supplying a different value for the restriction field) for the same ICS state:

1. IcsDownload(icsStateX, Normal | FAI, {**PidTagAssociated** (section 2.2.1.5) equals **FALSE**})

   => (diff1, icsStateA)

2. IcsDownload(icsStateA, Normal | FAI, {**PidTagAssociated** equals **TRUE**})

   => (diff2, icsStateB)

As a result, this sequence does not yield the same result:

- diff1 contains soft deletion notifications for any previously downloaded messaging objects mentioned in icsStateX. The **MetaTagIdsetGiven** property (section 2.2.1.1.1), which does not have a **PidTagAssociated** property value equals **FALSE**.

- diff2 contains soft deletions for all messaging objects mentioned in icsStateA.**MetaTagIdsetGiven**.

- icsStateB.**MetaTagIdsetGiven** only contains IDs of FAI messages.

3.3.5.6 Client Side Checkpointing

Checkpointing is a method of ICS state management that is used for upload and download operations to provide updated state information on an object-by-object basis. If a client has to abort synchronizations regularly, or if mitigating the effects of application or connection termination is a priority, the client can implement an ICS state checkpointing strategy. This is made possible by setting the **Eid** and **CN** flags in the **SynchronizationExtraFlags** field in the **ulExtra** field of the **RopSynchronizationConfigure** ROP request (section 2.2.3.2.1.1) that is made when initializing a download. As specified in section 3.2.5.9.1.1, setting the **Eid** and **CN** flags causes the server to send the **PidTagMid** property (section 2.2.1.2.1) and the **PidTagChangeNumber** property (section 2.2.1.2.3) in the ICS header for each messaging object. Using the **PidTagChangeNumber** property and the item's **PidTagMid** property, the ICS state can be maintained on the client by updating its **MetaTagIdsetGiven** (section 2.2.1.1.1) and **MetaTagCnsetSeen** (section 2.2.1.1.2) (or **MetaTagCnsetSeenFAI** (section 2.2.1.1.3), for folder associated information (FAI) messages) properties by using the **PidTagMid** and **PidTagChangeNumber** properties, respectively. Each of these properties is specified in section 2.2.1.1 or section 2.2.1.2.

If this updated ICS state is then persisted periodically, the client does not have to redownload all items in the event of a cancellation or other abnormal termination. It is recommended that the client persist the state downloaded at the end of the current (or a subsequent) ICS download process, after the download is complete. This is because the ICS state provided by the server is a much more efficient version than the version obtained by using checkpointing alone.

Note Checkpointing for synchronization download operations functions differently than checkpointing for synchronization upload operations. During a synchronization upload operation, the server returns checkpoint ICS states that are accurate to the time at which the checkpoint was requested. During a synchronization download operation, the server SHOULD<44> return the initial ICS state, or

MAY<45> return the checkpoint ICS state that is accurate to the time at which the checkpoint was requested, until the download is complete, at which time it returns the final ICS state.

3.3.5.7 Sending FastTransfer ROPs

3.3.5.7.1 Sending a RopFastTransferSourceGetBuffer ROP Request

To obtain all data output by an operation, the **RopFastTransferSourceGetBuffer** ROP (section 2.2.3.1.1.5) MUST be sent iteratively, because the amount of data that can be passed in one RPC is limited by its maximum size. A client MUST stop sending this ROP on a download context as soon as it receives **TransferStatus** field values of **Done** or **Error**.

Clients SHOULD<46> set the value of the **BufferSize** field in the ROP request to a sentinel value of 0xBABE to achieve maximum efficiency. If this field is not set to 0xBABE, then clients MUST pass a value equal to or greater than 15480 or MUST be prepared to increase this number in future requests if they passed a smaller value and the **RopBufferTooSmall** ROP response ([MS-OXCROPS] section 2.2.15.1) was returned.

Clients MAY set the value of the **MaximumBufferSize** field to at least the size of the output RPC buffer to achieve maximum efficiency.

For details about the client behavior when receiving a **RopFastTransferSourceGetBuffer** ROP response, see section 3.3.5.9.1.

3.3.5.7.2 Sending a RopTellVersion ROP Request

Clients MUST pass the server version exactly as it was obtained either from the **EcDoConnectEx** method response, as specified in [MS-OXCRPC], or from the **X-ServerApplication** header of the **Connect** request type response, as specified in [MS-OXCMAPIHTTP]. For more details about the only application scenario for this ROP, server-to-client-to-server upload, see section 3.3.4.2.1.

If the client sends the **RopTellVersion** ROP (section 2.2.3.1.1.6), the request MUST be sent before the first **RopFastTransferSourceGetBuffer** (section 2.2.3.1.1.5) or **RopFastTransferDestinationPutBuffer** ROP (section 2.2.3.1.2.2).

3.3.5.8 Sending ICS ROPs

This section specifies client requirements when sending ICS ROP requests to synchronize messaging objects with the server.

3.3.5.8.1 Sending a RopSynchronizationConfigure ROP Request

The client MUST upload the last remaining piece of configuration data, the initial ICS state, as specified in section 3.3.4.3.1, before it can request a FastTransfer stream that contains differences from the server.

For details about the client behavior when receiving a response to this ROP, see section 3.3.5.9.

3.3.5.8.2 Sending a RopSynchronizationUploadStateStreamBegin ROP Request

When the **RopSynchronizationUploadStateStreamBegin** ROP request (section 2.2.3.2.2.1) is sent to the server, no other property upload MUST be in progress for this synchronization context, and any property that is being specified in this ROP SHOULD NOT have been previously uploaded into this synchronization context.

This ROP MUST be followed by the **RopSynchronizationUploadStateStreamContinue** ROP (section 2.2.3.2.2.2) or the **RopSynchronizationUploadStateStreamEnd** ROP (section 2.2.3.2.2.3).

3.3.5.8.3 Sending a RopSynchronizationUploadStateStreamContinue ROP Request

This ROP MUST be followed by the **RopSynchronizationUploadStateStreamContinue** ROP (section 2.2.3.2.2.2) or the **RopSynchronizationUploadStateStreamEnd** ROP (section 2.2.3.2.2.3).

Uploading the ICS state properties MUST be initiated by sending the **RopSynchronizationUploadStateStreamBegin** ROP (section 2.2.3.2.2.1).

Clients SHOULD skip this ROP if the size of the remaining data specified in the **StreamDataSize** field is 0.

3.3.5.8.4 Sending a RopSynchronizationUploadStateStreamEnd ROP Request

Uploading the ICS state properties MUST be initiated by sending the **RopSynchronizationUploadStateStreamBegin** ROP request (section 2.2.3.2.2.1) followed by zero or more iterations of the **RopSynchronizationUploadStateStreamContinue** ROP (section 2.2.3.2.2.2).

3.3.5.8.5 Sending a RopSynchronizationGetTransferState ROP Request

Clients are only required to use the **RopSynchronizationGetTransferState** ROP (section 2.2.3.2.3.1) when performing synchronization uploads, as it is the only way to obtain the ICS state maintained on the synchronization upload context.

For details about the client behavior when receiving a response to this ROP, see section 3.3.5.9.

3.3.5.8.6 Sending a RopSynchronizationOpenCollector ROP Request

A client SHOULD upload the initial ICS state, as specified in section 2.2.3.2.2, into the synchronization context returned in the **RopSynchronizationOpenCollector** ROP (section 2.2.3.2.4.1) response prior to using any **RopSynchronizationImport*** ROPs. However, the client can elect not to upload the initial ICS state. For details about how that would affect the responsibilities of the protocol roles, see section 3.3.4.3.3.

Be sure to update the stored **MetaTagIdsetGiven** property (section 2.2.1.1.1) value with internal identifiers of the objects that were imported into the server replica. These identifiers either are returned in the **RopSynchronizationImport*** ROP responses or can be extracted from **GID** structures ([MS-OXCDATA] section 2.2.1.3) sent as the input **PidTagSourceKey** property (section 2.2.1.2.5) values.

For details about the client behavior when receiving a response to this ROP, see section 3.3.5.9.

3.3.5.8.7 Sending a RopSynchronizationImportMessageChange ROP Request

When uploading new messages, clients SHOULD add their **Message ID** structures ([MS-OXCDATA] section 2.2.1.2) to the **MetaTagIdsetGiven** property (section 2.2.1.1.1) value upon successful completion of this ROP.

Note that because a server returns an empty message from the **RopSynchronizationImportMessageChange** ROP ([MS-OXCROPS] section 2.2.13.2), even when uploading changes to an existing message, this ROP can only be used to perform upload of full message changes or new messages. In order for the client to upload partial message changes, it SHOULD take them outside the synchronization upload operation, by initiating an upload by using the **RopOpenMessage** ROP ([MS-OXCROPS] section 2.2.6.1) followed by other ROPs discussed in [MS-OXCMSG], such as the **RopSetProperties** ROP ([MS-OXCROPS] section 2.2.8.6) and the **RopModifyRecipients** ROP ([MS-OXCROPS] section 2.2.6.5). However, these ROPs do not let the client set values to any of the properties that the **RopSynchronizationImportMessageChange** ROP accepts.

The **RopSynchronizationImportMessageChange** ROP returns the handle of a Message object, which the client MUST populate with the contents of the message. The client populates the Message object by sending the **ROPSetProperties** ROP, the **ROPCreateAttachment** ROP ([MS-OXCROPS] section 2.2.6.13), and other ROPs required to populate the message contents, as specified in [MS-OXCMSG] section 3.1.4, followed by the **ROPSaveChangesMessage** ROP ([MS-OXCROPS] section 2.2.6.3).

For details about the client behavior when receiving a response to this ROP, see section 3.3.5.9.

3.3.5.8.8 Sending a RopSynchronizationImportHierarchyChange ROP Request

When uploading new folders, clients SHOULD update the ICS state that corresponds to the chosen synchronization scope by adding the **Folder ID** structures ([MS-OXCDATA] section 2.2.1.1) of new folders to the **MetaTagIdsetGiven** property (section 2.2.1.1.1) upon successful completion of this ROP.

Changes to parent folders MUST be made before changes to child folders. For example, the client cannot send the **RopSynchronizationImportHierarchyChange** ROP (section 2.2.3.2.4.3) with a subfolder change before informing the server of the existence of the parent folder.

To move a folder to a different subfolder within the same private mailbox, the client MUST pass the **PidTagSourceKey** property (section 2.2.1.2.5) of a destination parent folder in the **PidTagParentSourceKey** property (section 2.2.1.2.6) in the **HierarchyValues** field while passing the value of the **PidTagSourceKey** property of the folder being moved in the **PidTagSourceKey** property. Moving folders within a public mailbox is not supported.

3.3.5.8.9 Sending a RopSynchronizationImportMessageMove ROP Request

When uploading new messages, clients SHOULD update the ICS state of the source folder by removing the **Message ID** structure ([MS-OXCDATA] section 2.2.1.2) of moved the messages from its **MetaTagIdsetGiven** property (section 2.2.1.1.1). Otherwise, the client MUST be prepared to receive deletion notifications for these messages in the source folder during the next ICS download.

Clients MUST only pass folders from private mailboxes in the **InputServerObject** field.

3.3.5.8.10 Sending a RopSynchronizationImportDeletes ROP Request

Clients SHOULD update the ICS state of the chosen synchronization scope by removing internal identifiers of deleted objects from its **MetaTagIdsetGiven** property (section 2.2.1.1.1). Otherwise, the client will receive deletion notifications for these messages during the next ICS download.

Clients SHOULD expect this ROP to fail if deletion of any of the objects passed in the request buffer fails, except for the common cases specified in section 2.2.3.2.4.5. The possibility of a failure is higher when the user has lower privileges to a mailbox—this is especially a consideration for delegate and public folder access. It is recommended that clients that use this ROP have a strategy to retry this operation, which can be a combination of the following steps:

1. Retry the ROP with the same arguments on a new synchronization upload context.
2. Retry the ROP, passing one ID at a time.
3. Retry the ROP by using online mode ROPs, such as **RopDeleteFolder** and **RopDeleteMessages**, as specified in [MS-OXCFOLD] section 2.2.1.3 and 2.2.1.11, respectively.
4. Perform the ICS download, resolving server changes against their own pending synchronization upload context.
5. Skip an object and undo the operation in the local replica.

3.3.5.8.11 Sending a RopSynchronizationImportReadStateChanges ROP Request

Clients SHOULD expect this ROP to fail if any read state changes on the objects passed in the request buffer fail. The possibility of a failure is higher when the user has lower privileges to a mailbox; this is especially a consideration for delegate and public folder access. Clients that use this ROP SHOULD have a strategy to retry this operation, which can be a combination of the following steps:

- Retry the ROP with the same arguments on a new synchronization upload context.
- Retry the ROP, passing one ID at a time.
- Retry the ROP by using online mode ROPs, such as the **RopSetMessageReadFlag** ROP ([MS-OXCROPS] section 2.2.6.11).
- Perform the ICS download, resolving server changes against their own pending synchronization upload context.
- Skip an object and undo the operation in the local replica.

3.3.5.8.12 Sending a RopGetLocalReplicaIds ROP Request

Clients SHOULD NOT allocate another batch of IDs until the one they allocated before is used up. Allocating IDs in batches of moderate size, between 0x00000200 and 0x0000FFFF, is recommended. Note that servers are responsible for enforcing restrictions on the number of IDs that can be allocated at one time.

The client can reconstruct all allocated **GID** structures ([MS-OXCDATA] section 2.2.1.3) by combining the returned **REPLGUID** structure with any **GLOBCNT** structure values, as specified in section 2.2.2.5, from the [GlobalCount, GlobalCount + IdCount – 1] range.

The client SHOULD use the obtained IDs whenever creating new folders or new messages in any folder within its local replica. For more details about how clients can assign identifiers to objects created in a local replica, see section 3.3.5.2.

3.3.5.8.13 Sending a RopSetLocalReplicaMidsetDeleted ROP Request

The client MUST ensure that ranges supplied as request fields to the **RopSetLocalReplicaMidsetDeleted** ROP (section 2.2.3.2.4.8) are allocated by using the **RopGetLocalReplicaIds** ROP (section 2.2.3.2.4.7).

The value of the **LongTermIdRanges** field MUST only identify IDs that were assigned to an item on the client but never uploaded to the server, as well as IDs that the client will never use in the future. The value of the **LongTermIdRanges** field MUST NOT identify IDs that exist on the server; failure to comply with this requirement results in an inconsistent state between the client and server as the server adds the ID to the deleted items list and the item will be deleted from the client during the next synchronization while the item will still exist on the server.

The following example shows a possible implementation of the client with regards to assignment of server-allocated IDs, as specified in section 3.3.5.2.1, to objects in a local replica. Clients do not have to follow the example specified in this section; it is only used to show the applicability of the **RopSetLocalReplicaMidsetDeleted** ROP (section 2.2.3.2.4.8):

1. Initially, a client has no server-allocated IDs that it can assign to objects that are created when working offline, so it is required to ask the server to allocate a block of IDs by sending the **RopGetLocalReplicaIds** ROP (section 2.2.3.2.4.7). The server responds with a block of IDs that the client stores in a local replica.
2. The client requires the server-allocated ID whenever it has to create a message in a folder in a local replica. For that purpose, the client associates a range of IDs previously allocated with the

RopGetLocalReplicaIds ROP with a folder, so that IDs from that range can be used for new or moved items in that folder.

3. The client creates a message in the local replica and assigns it a server-allocated ID from the set of IDs previously allocated to the folder from a call to the **RopGetLocalReplicaIds** ROP in step 2.

4. The client then deletes the message from the local replica before the message is uploaded to the server, because, for example, the client is offline.

5. The client issues the **RopSetLocalReplicaMidsetDeleted** ROP for the ID that was consumed by the client, but never passed to the server.

3.3.5.9 Receiving FastTransfer and ICS ROP Responses

The object output in the **OutputServerObject** field MUST be released using the **RopRelease** ROP ([MS-OXCROPS] section 2.2.15.3) as soon as the client no longer requires it.

3.3.5.9.1 Receiving a RopFastTransferSourceGetBuffer ROP Response

The FastTransfer stream on download is read-only and non-seekable, and is usually generated by the server when requested by the client. Once it is obtained, data cannot be requeried, unless the operation is reconfigured from the beginning. Even then, there is no guarantee that the content of the stream will be the same as during the previous attempt.

As streams can be very large, clients SHOULD decode portions of the FastTransfer stream as they arrive in the **RopFastTransferSourceGetBuffer** ROP response buffers.

Clients that have not performed version detection on the server and have leveraged the server's flexibility to ignore unknown flags in the **SynchronizationExtraFlags** field of the **RopSynchronizationConfigure** ROP ([MS-OXCROPS] section 2.2.13.1) MUST parse the FastTransfer stream to detect whether the server honored each of the requested behavior flags. This enables the client to request behavior from an older server that does not support newer behaviors without having to strictly check the version of the server.

If the client receives a value of 0x00000480 in the **ReturnValue** field, the client SHOULD<47> wait at least the period of time specified in **BackoffTime** before retrying the ROP. The complete list of error codes is specified in [MS-OXCDATA] section 2.4.

If a cancellation occurs on a download and there is any data left unprocessed from the result of the **RopFastTransferSourceGetBuffer** ROP ([MS-OXCROPS] section 2.2.12.3), the client SHOULD NOT update the ICS state by using the **RopSynchronizationGetTransferState** ROP ([MS-OXCROPS] section 2.2.13.8). This is because the server does not know if the client successfully committed all items. It is possible to miss items if the **RopSynchronizationGetTransferState** ROP is used at a time when synchronization buffers are only partially processed.

3.3.5.10 Client Specific Handling

Clients can choose to handle pieces of the data stream in specific ways to improve the efficiency or user experience. Examples of this are progress information, as specified in section 2.2.2.7, and error information, as specified by the **ExtendedErrorInfo** structure in section 2.2.2.10, and the **FxErrorInfo** marker in section 2.2.4.1.4. A possible implementation uses progress information to compute and inform the end user how much time is left in the ICS process. Another possible implementation to handle errors is to use the **Folder ID**, as specified in [MS-OXCDATA] section 2.2.1.1, and **Message ID**, as specified in [MS-OXCDATA] section 2.2.1.2, following the **FxErrorInfo** marker and move the items into a folder that is not synchronized. By moving the items to another folder, the error does not continue to occur during each subsequent synchronization. Error information can be used to move the problem items to another folder to avoid repeated errors during the next ICS upload or download.

3.3.5.11 Client Conflict Resolution

A client SHOULD avoid conflicts by detecting them and trying to run logic to resolve the conflict.

A possible implementation of client conflict detection is to use the **RopOpenMessage** ROP ([MS-OXCROPS] section 2.2.6.1) to open the message and then call the **RopGetPropertiesSpecific** ROP ([MS-OXCROPS] section 2.2.8.3) to retrieve the message state properties (**PidTagChangeKey** (section 2.2.1.2.7), **PidTagPredecessorChangeList** (section 2.2.1.2.8), and **PidTagLastModificationTime** ([MS-OXCMSG] section 2.2.2.2)). The values of the properties are then compared using the logic specified in section 3.1.5.6.1 to determine whether the message is different on the server and whether the message is in conflict. If the messages are the same, the client can apply the local change and save the message.

If the client detects a conflict, the client can use standard message and folder ROPs, such as the **RopGetPropertiesSpecific** ROP ([MS-OXCROPS] section 2.2.8.3), the **RopSetProperties** ROP ([MS-OXCROPS] section 2.2.8.6), the **RopSetReadFlags** ROP ([MS-OXCROPS] section 2.2.6.10), and the **RopDeleteProperties** ROP ([MS-OXCROPS] section 2.2.8.8), to apply changes to the server replica, therefore making it a nonconflicting version.

Then, regardless of whether the client detected a conflict or not, the client then modifies the local client item and local client ICS state in a way that would either prevent the item from being downloaded (local wins), or force (resolved or server wins) the item to be downloaded, as specified in section 3.1.5.6. The client then calls **RopSaveMessageChanges** ROP ([MS-OXCROPS] section 2.2.6.3) requesting that the message remain open following the save, and then calls **RopGetPropertiesSpecific** ROP to retrieve the new **PidTagChangeKey** (section 2.2.1.2.7), **PidTagLastModificationTime** ([MS-OXCMSG] section 2.2.2.2) and **PidTagPredecessorChangeList** (section 2.2.1.2.8) property. The client then updates the message state properties on the local item and modifies the ICS state properties to reflect those changes.

When a client is unable to automatically resolve all conflicting changes, a possible implementation would preserve alternate versions of messages, which can be made accessible to the user in case they prefer the alternate version.

A client can, for example, pick a "winning" message based on the **PidTagLastModificationTime** property ([MS-OXCMSG] section 2.2.2.2) and leave this in place of the modified item. The other version of the message, deemed the "loser," can be moved to a folder that contains previous versions of conflicting messages.

3.3.5.12 Using the PidTagMessageSize Property Value

The value for the **PidTagMessageSize** property MUST be treated only as an estimate by the client. For more details, see section 3.2.5.4.

3.3.5.13 Sending the MetaTagIdsetGiven ICS State Property

The property tag for this property suggests that it is of type **PtypInteger32** ([MS-OXCDATA] section 2.11.1), but the data MUST be handled as **PtypBinary** ([MS-OXCDATA] section 2.11.1) data by clients. Clients SHOULD send the **MetaTagIdsetGiven** property (section 2.2.1.1.1) with a property tag that defines it as **PtypInteger32**.

This property is not downloaded back to the client in the final ICS state obtained for them through the **RopSynchronizationGetTransferState** ROP (section 2.2.3.2.3.1). Clients SHOULD remove this property before uploading the initial ICS state on synchronization upload contexts and clients MUST merge this property back in when receiving the final ICS state from the server. Clients MUST add IDs of messaging objects created in or originating from a local replica to this property by using a process called checkpointing, as specified in section 3.3.5.6.

3.3.6 Timer Events

None.

3.3.7 Other Local Events

None.

4 Protocol Examples

4.1 Hierarchy Synchronization Examples

4.1.1 Adding or Modifying a Folder

The following example shows the ROPs involved in synchronizing changes between the client and server when adding or modifying a folder. The user has previously created or modified a folder on both the client and the server and has just connected to the server to synchronize the changes.

1. **RopLogon** ROP ([MS-OXCROPS] section 2.2.3.1) – The server returns the ID of the **interpersonal messaging subtree** folder in this call.
2. **RopOpenFolder** ROP ([MS-OXCROPS] section 2.2.4.1) – Open the interpersonal messaging subtree folder.
3. **RopSynchronizationOpenCollector** ROP (section 2.2.3.2.4.1) – Open the hierarchy synchronization upload context by using the handle of the interpersonal messaging subtree folder.
4. **RopSynchronizationUploadStateStreamBegin** ROP (section 2.2.3.2.2.1) – Upload the ICS state property **MetaTagCnsetSeen** (section 2.2.1.1.2) by using the synchronization upload context.
5. **RopSynchronizationUploadStateStreamContinue** ROP (section 2.2.3.2.2.2) – Upload the ICS state property **MetaTagCnsetSeen** by using the synchronization upload context.
6. **RopSynchronizationUploadStateStreamEnd** ROP (section 2.2.3.2.2.3) – Upload the ICS state property **MetaTagCnsetSeen** by using the synchronization upload context.
7. **RopSynchronizationImportHierarchyChange** ROP (section 2.2.3.2.4.3) – Send the folder properties by using the synchronization upload context.
8. **RopSynchronizationGetTransferState** ROP (section 2.2.3.2.3.1) – Get the updated ICS state by using the synchronization upload context. This call returns a handle to a synchronization download context.
9. **RopFastTransferSourceGetBuffer** ROP (section 2.2.3.1.1.5) – Retrieve the ICS state data by using the synchronization download context.
10. **RopRelease** ROP ([MS-OXCROPS] section 2.2.15.3) – Release the synchronization download context.
11. **RopRelease** – Release the synchronization upload context.
12. **RopSynchronizationConfigure** ROP (section 2.2.3.2.1.1) – Open the hierarchy synchronization download context by using the handle of the interpersonal messaging subtree folder.
13. **RopSynchronizationUploadStateStreamBegin** ROP – Upload the ICS state property **MetaTagIdsetGiven** (section 2.2.1.1.1) by using the synchronization download context.
14. **RopSynchronizationUploadStateStreamContinue** ROP – Upload the ICS state property **MetaTagIdsetGiven** by using the synchronization download context.
15. **RopSynchronizationUploadStateStreamEnd** ROP – Upload the ICS state property **MetaTagIdsetGiven** by using the synchronization download context.
16. **RopSynchronizationUploadStateStreamBegin** ROP – Upload the ICS state property **MetaTagCnsetSeen** by using the synchronization download context.

17. **RopSynchronizationUploadStateStreamContinue** ROP – Upload the ICS state property **MetaTagCnsetSeen** by using the synchronization download context.

18. **RopSynchronizationUploadStateStreamEnd** ROP – Upload the ICS state property **MetaTagCnsetSeen** by using the synchronization download context.

19. **RopFastTransferSourceGetBuffer** ROP – Receive the folder properties and updated ICS state by using the synchronization download context.

20. **RopRelease** ROP – Release the synchronization download context.

21. **RopRelease** ROP – Release the interpersonal messaging subtree folder.

22. **RopRelease** ROP – Release the message store.

4.1.2 Deleting a Folder

The following example shows the ROPs involved in synchronizing changes between the client and server when deleting a folder. The user has previously deleted a folder on both the client and the server, and has just connected to the server to synchronize the changes.

1. **RopLogon** ROP ([MS-OXCROPS] section 2.2.3.1) – The server returns the ID of the interpersonal messaging subtree folder in this call.

2. **RopOpenFolder** ROP ([MS-OXCROPS] section 2.2.4.1) – Open the interpersonal messaging subtree folder.

3. **RopSynchronizationOpenCollector** ROP (section 2.2.3.2.4.1) – Open the hierarchy synchronization upload context by using the handle of the interpersonal messaging subtree folder.

4. **RopSynchronizationUploadStateStreamBegin** ROP (section 2.2.3.2.2.1) – Upload the ICS state property **MetaTagCnsetSeen** (section 2.2.1.1.2) by using the synchronization upload context.

5. **RopSynchronizationUploadStateStreamContinue** ROP (section 2.2.3.2.2.2) – Upload the ICS state property **MetaTagCnsetSeen** by using the synchronization upload context.

6. **RopSynchronizationUploadStateStreamEnd** ROP (section 2.2.3.2.2.2) – Upload the ICS state property **MetaTagCnsetSeen** by using the synchronization upload context.

7. **RopSynchronizationImportDeletes** ROP (section 2.2.3.2.4.5) – Send the information about the deleted folder by using the synchronization upload context.

8. **RopSynchronizationGetTransferState** ROP (section 2.2.3.2.3.1) – Get the updated ICS state by using the synchronization upload context. This call returns a handle to a synchronization download context.

9. **RopFastTransferSourceGetBuffer** ROP (section 2.2.3.1.1.5) – Retrieve the ICS state data by using the synchronization download context.

10. **RopRelease** ROP ([MS-OXCROPS] section 2.2.15.3) – Release the synchronization download context.

11. **RopRelease** ROP – Release the synchronization upload context.

12. **RopSynchronizationConfigure** ROP (section 2.2.3.2.1.1) – Open the hierarchy synchronization download context by using the handle of the interpersonal messaging subtree folder.

13. **RopSynchronizationUploadStateStreamBegin** ROP – Upload the ICS state property **MetaTagIdsetGiven** (section 2.2.1.1.1) by using the synchronization download context.

14. **RopSynchronizationUploadStateStreamContinue** ROP – Upload the ICS state property **MetaTagIdsetGiven** by using the synchronization download context.

15. **RopSynchronizationUploadStateStreamEnd** ROP – Upload the ICS state property **MetaTagIdsetGiven** by using the synchronization download context.

16. **RopSynchronizationUploadStateStreamBegin** ROP – Upload the ICS state property **MetaTagCnsetSeen** by using the synchronization download context.

17. **RopSynchronizationUploadStateStreamContinue** ROP – Upload the ICS state property **MetaTagCnsetSeen** by using the synchronization download context.

18. **RopSynchronizationUploadStateStreamEnd** ROP – Upload the ICS state property **MetaTagCnsetSeen** by using the synchronization download context.

19. **RopFastTransferSourceGetBuffer** ROP – Receive the information about the deleted folder and updated ICS state by using the synchronization download context.

20. **RopRelease** ROP – Release the synchronization download context.

21. **RopRelease** ROP – Release the interpersonal messaging subtree folder.

22. **RopRelease** ROP – Release the message store.

4.2 Message Synchronization Upload Examples

4.2.1 Creating or Modifying a Message

The following example shows the ROPs involved in synchronizing changes between the client and server when creating or modifying a message. The user has previously created or modified a message on both the client and the server, and has just connected to the server to synchronize the changes.

1. **RopLogon** ROP ([MS-OXCROPS] section 2.2.3.1) – Open the message store.

2. **RopOpenFolder** ROP ([MS-OXCROPS] section 2.2.4.1) – Open the folder being synchronized.

3. **RopSynchronizationOpenCollector** ROP (section 2.2.3.2.4.1) – Open the content synchronization upload context by using the handle of the folder being synchronized.

4. **RopSynchronizationUploadStateStreamBegin** ROP (section 2.2.3.2.4.1) – Upload the ICS state property **MetaTagCnsetSeen** (section 2.2.1.1.2) by using the synchronization upload context.

5. **RopSynchronizationUploadStateStreamContinue** ROP (section 2.2.3.2.4.1) – Upload the ICS state property **MetaTagCnsetSeen** by using the synchronization upload context.

6. **RopSynchronizationUploadStateStreamEnd** ROP (section 2.2.3.2.4.1) – Upload the ICS state property **MetaTagCnsetSeen** by using the synchronization upload context.

7. **RopSynchronizationUploadStateStreamBegin** ROP – Upload the ICS state property **MetaTagCnsetSeenFAI** (section 2.2.1.1.3) by using the synchronization upload context.

8. **RopSynchronizationUploadStateStreamContinue** ROP – Upload the ICS state property **MetaTagCnsetSeenFAI** by using the synchronization upload context.

9. **RopSynchronizationUploadStateStreamEnd** ROP – Upload the ICS state property **MetaTagCnsetSeenFAI** by using the synchronization upload context.

10. **RopSynchronizationUploadStateStreamBegin** ROP – Upload the ICS state property **MetaTagCnsetRead** (section 2.2.1.1.4) by using the synchronization upload context.

11. **RopSynchronizationUploadStateStreamContinue** ROP – Upload the ICS state property **MetaTagCnsetRead** by using the synchronization upload context.

12. **RopSynchronizationUploadStateStreamEnd** ROP – Upload the ICS state property **MetaTagCnsetRead** by using the synchronization upload context.

13. **RopSynchronizationImportMessageChange** ROP (section 2.2.3.2.4.2) – Acquire a Message object (with a specified ID) by using the synchronization upload context. If the message does not yet exist, it will be created. This call returns a handle to a Message object.

14. **RopSetProperties** ROP ([MS-OXCROPS] section 2.2.8.6) – Set the message properties by using the message handle.

15. **RopSaveChangesMessage** ROP ([MS-OXCROPS] section 2.2.6.3) – Save the message by using the message handle.

16. **RopRelease** ROP ([MS-OXCROPS] section 2.2.15.3) – Release the message.

17. **RopSynchronizationGetTransferState** ROP (section 2.2.3.2.3.1) – Get the updated ICS state by using the upload context. This call returns a handle to a synchronization download context.

18. **RopFastTransferSourceGetBuffer** ROP (section 2.2.3.2.3.1) – Retrieve the ICS state data by using the synchronization download context.

19. **RopRelease** ROP – Release the synchronization download context.

20. **RopRelease** ROP – Release the synchronization upload context.

21. **RopSynchronizationConfigure** ROP (section 2.2.3.2.3.1) – Open the content synchronization download context by using the handle of the folder being synchronized.

22. **RopSynchronizationUploadStateStreamBegin** ROP – Upload the ICS state property **MetaTagIdsetGiven** (section 2.2.1.1.1) by using the synchronization download context.

23. **RopSynchronizationUploadStateStreamContinue** ROP – Upload the ICS state property **MetaTagIdsetGiven** by using the synchronization download context.

24. **RopSynchronizationUploadStateStreamEnd** ROP – Upload the ICS state property **MetaTagIdsetGiven** by using the synchronization download context.

25. **RopSynchronizationUploadStateStreamBegin** ROP – Upload the ICS state property **MetaTagCnsetSeen** by using the synchronization download context.

26. **RopSynchronizationUploadStateStreamContinue** ROP – Upload the ICS state property **MetaTagCnsetSeen** by using the synchronization download context.

27. **RopSynchronizationUploadStateStreamEnd** ROP – Upload the ICS state property **MetaTagCnsetSeen** by using the synchronization download context.

28. **RopSynchronizationUploadStateStreamBegin** ROP – Upload the ICS state property **MetaTagCnsetSeenFAI** by using the synchronization download context.

29. **RopSynchronizationUploadStateStreamContinue** ROP – Upload the ICS state property **MetaTagCnsetSeenFAI** by using the synchronization download context.

30. **RopSynchronizationUploadStateStreamEnd** ROP – Upload the ICS state property **MetaTagCnsetSeenFAI** by using the synchronization download context.

31. **RopSynchronizationUploadStateStreamBegin** ROP – Upload the ICS state property **MetaTagCnsetRead** by using the synchronization download context.

32. **RopSynchronizationUploadStateStreamContinue** ROP – Upload the ICS state property **MetaTagCnsetRead** by using the synchronization download context.

33. **RopSynchronizationUploadStateStreamEnd** ROP – Upload the ICS state property **MetaTagCnsetRead** by using the synchronization download context.

34. **RopFastTransferSourceGetBuffer** ROP – Receive the folder properties and updated ICS state by using the synchronization download context.

35. **RopRelease** ROP – Release the synchronization download context.

36. **RopRelease** ROP – Release the folder.

37. **RopRelease** ROP – Release the message store.

4.2.2 Deleting a Message

The following example shows the ROPs involved in synchronizing changes between the client and server when deleting a message. The user has previously deleted a message on both the client and the server and has just connected to the server to synchronize the changes.

1. **RopLogon** ROP ([MS-OXCROPS] section 2.2.3.1) – Open the message store.

2. **RopOpenFolder** ROP ([MS-OXCFOLD] section 2.2.1.1) – Open the folder that is being synchronized.

3. **RopSynchronizationOpenCollector** ROP (section 2.2.3.2.4.1) – Open the content synchronization upload context by using the handle of the folder that is being synchronized.

4. **RopSynchronizationUploadStateStreamBegin** ROP (section 2.2.3.2.2.1) – Upload the ICS state property **MetaTagCnsetSeen** (section 2.2.3.2.1.1) by using the synchronization upload context.

5. **RopSynchronizationUploadStateStreamContinue** ROP (section 2.2.3.2.2.2) – Upload the ICS state property **MetaTagCnsetSeen** by using the synchronization upload context.

6. **RopSynchronizationUploadStateStreamEnd** ROP (section 2.2.3.2.2.3) – Upload the ICS state property **MetaTagCnsetSeen** by using the synchronization upload context.

7. **RopSynchronizationUploadStateStreamBegin** ROP – Upload the ICS state property **MetaTagCnsetSeenFAI** (section 2.2.1.1.3) by using the synchronization upload context.

8. **RopSynchronizationUploadStateStreamContinue** ROP – Upload the ICS state property **MetaTagCnsetSeenFAI** by using the synchronization upload context.

9. **RopSynchronizationUploadStateStreamEnd** ROP – Upload the ICS state property **MetaTagCnsetSeenFAI** by using the synchronization upload context.

10. **RopSynchronizationUploadStateStreamBegin** ROP – Upload the ICS state property **MetaTagCnsetRead** (section 2.2.1.1.4) by using the synchronization upload context.

11. **RopSynchronizationUploadStateStreamContinue** ROP – Upload the ICS state property **MetaTagCnsetRead** by using the synchronization upload context.

12. **RopSynchronizationUploadStateStreamEnd** ROP – Upload the ICS state property **MetaTagCnsetRead** by using the synchronization upload context.

13. **RopSynchronizationImportDeletes** ROP (section 2.2.3.2.4.5) – Send the information about the deleted message by using the synchronization upload context.

14. **RopSynchronizationGetTransferState** ROP (section 2.2.3.2.3.1) – Get the updated ICS state by using the synchronization upload context. This call returns a handle to a synchronization download context.

15. **RopFastTransferSourceGetBuffer** ROP (section 2.2.3.1.1.5) – Retrieve the ICS state data by using the synchronization download context.

16. **RopRelease** ROP ([MS-OXCROPS] section 2.2.15.3) – Release the synchronization download context.

17. **RopRelease** ROP – Release the synchronization upload context.

18. **RopSynchronizationConfigure** (section 2.2.3.2.1.1) – Open the content synchronization download context by using the handle of the folder that is being synchronized.

19. **RopSynchronizationUploadStateStreamBegin** ROP – Upload the ICS state property **MetaTagIdsetGiven** (section 2.2.1.1.1) by using the synchronization download context.

20. **RopSynchronizationUploadStateStreamContinue** ROP – Upload the ICS state property **MetaTagIdsetGiven** by using the synchronization download context.

21. **RopSynchronizationUploadStateStreamEnd** ROP – Upload the ICS state property **MetaTagIdsetGiven** by using the synchronization download context.

22. **RopSynchronizationUploadStateStreamBegin** ROP – Upload the ICS state property **MetaTagCnsetSeen** by using the synchronization download context.

23. **RopSynchronizationUploadStateStreamContinue** ROP – Upload the ICS state property **MetaTagCnsetSeen** by using the synchronization download context.

24. **RopSynchronizationUploadStateStreamEnd** ROP – Upload the ICS state property **MetaTagCnsetSeen** by using the synchronization download context.

25. **RopSynchronizationUploadStateStreamBegin** ROP – Upload the ICS state property **MetaTagCnsetSeenFAI** by using the synchronization download context.

26. **RopSynchronizationUploadStateStreamContinue** ROP – Upload the ICS state property **MetaTagCnsetSeenFAI** by using the synchronization download context.

27. **RopSynchronizationUploadStateStreamEnd** ROP – Upload the ICS state property **MetaTagCnsetSeenFAI** by using the synchronization download context.

28. **RopSynchronizationUploadStateStreamBegin** ROP – Upload the ICS state property **MetaTagCnsetRead** by using the synchronization download context.

29. **RopSynchronizationUploadStateStreamContinue** ROP – Upload the ICS state property **MetaTagCnsetRead** by using the synchronization download context.

30. **RopSynchronizationUploadStateStreamEnd** ROP – Upload the ICS state property **MetaTagCnsetRead** by using the synchronization download context.

31. **RopFastTransferSourceGetBuffer** ROP – Receive the information about the deleted message and updated ICS state by using the synchronization download context.

32. **RopRelease** ROP – Release the synchronization download context.

33. **RopRelease** ROP – Release the folder.

34. **RopRelease** ROP – Release the message store.

4.3 Partial Item Examples

4.3.1 Uploading a Partial Item

The following example shows the ROPs involved in synchronizing changes between the client and server when uploading a partial message to the server. The user has previously modified a message on the client and has just connected to the server to synchronize the change. This partial upload occurs without using the **RopSynchronizationImportMessageChange** ROP (section 2.2.3.2.4.2).

1. **RopLogon** ROP ([MS-OXCSTOR] section 2.2.1.1) – Open the message store.
2. **RopOpenFolder** ROP ([MS-OXCFOLD] section 2.2.1.1) – Open the folder being synchronized.
3. **RopSynchronizationOpenCollector** ROP (section 2.2.3.2.4.1) – Open the content synchronization upload context by using the handle of the folder being synchronized.
4. **RopSynchronizationUploadStateStreamBegin** ROP (section 2.2.3.2.2.1) – Upload the ICS state property **MetaTagCnsetSeen** ROP (section 2.2.1.1.2) by using the synchronization upload context.
5. **RopSynchronizationUploadStateStreamContinue** ROP (section 2.2.3.2.2.1) – Upload the ICS state property **MetaTagCnsetSeen** by using the synchronization upload context.
6. **RopSynchronizationUploadStateStreamEnd** ROP (section 2.2.3.2.2.3) – Upload the ICS state property **MetaTagCnsetSeen** by using the synchronization upload context.
7. **RopSynchronizationUploadStateStreamBegin** ROP – Upload the ICS state property **MetaTagCnsetSeenFAI** (section 2.2.1.1.3) by using the synchronization upload context.
8. **RopSynchronizationUploadStateStreamContinue** ROP – Upload the ICS state property **MetaTagCnsetSeenFAI** by using the synchronization upload context.
9. **RopSynchronizationUploadStateStreamEnd** ROP – Upload the ICS state property **MetaTagCnsetSeenFAI** by using the synchronization upload context.
10. **RopSynchronizationUploadStateStreamBegin** ROP – Upload the ICS state property **MetaTagCnsetRead** (section 2.2.1.1.4) by using the synchronization upload context.
11. **RopSynchronizationUploadStateStreamContinue** ROP – Upload the ICS state property **MetaTagCnsetRead** by using the synchronization upload context.
12. **RopSynchronizationUploadStateStreamEnd** ROP – Upload the ICS state property **MetaTagCnsetRead** by using the synchronization upload context.
13. **RopOpenMessage** ROP ([MS-OXCMSG] section 2.2.3.1) – Acquire a Message object with a specified ID. This call returns a handle to a Message object.
14. **RopGetPropertiesSpecific** ROP ([MS-OXCPRPT] section 2.2.2) – Get the values of **PidTagPredecessorChangeList** (section 2.2.1.2.8) and **PidTagChangeKey** (section 2.2.1.2.7) properties by using the message handle.
15. **RopDeletePropertiesNoReplicate** ROP ([MS-OXCPRPT] section 2.2.8) – Delete properties that were deleted on the local message.
16. **RopSetProperties** ROP ([MS-OXCPRPT] section 2.2.5) – Set new/updated message properties by using the message handle. In addition, also set an updated value of the **PidTagPredecessorChangeList** property to avoid having this change redownloaded to the client.
17. **RopSaveChangesMessage** ROP ([MS-OXCMSG] section 2.2.3.3) – Save the message by using the message handle.

18. **RopRelease** ROP ([MS-OXCROPS] section 2.2.15.3) – Release the message.

19. **RopSynchronizationGetTransferState** ROP (section 2.2.3.2.3.1) – Get the updated ICS state by using the synchronization upload context. This call returns a handle to a synchronization download context.

20. **RopFastTransferSourceGetBuffer** ROP (section 2.2.3.1.1.5) – Retrieve the ICS state data by using the synchronization download context.

21. **RopRelease** ROP – Release the synchronization download context.

22. **RopRelease** ROP – Release the synchronization upload context.

23. **RopRelease** ROP – Release the folder.

24. **RopRelease** ROP – Release the message store.

4.3.2 Downloading a Partial Item

The following example shows the ROPs involved in synchronizing changes between the client and server when downloading a partial message to the client. The user has previously modified a message on the server and has just connected to the server to synchronize the changes by using the **PartialItem** download flag.

1. **RopLogon** ROP ([MS-OXCSTOR] section 2.2.1.1) – Open the message store.

2. **RopOpenFolder** ROP ([MS-OXCROPS] section 2.2.4.1) – Open the folder being synchronized.

3. **RopSynchronizationConfigure** ROP (section 2.2.3.2.1.1) – Open the content synchronization download context by using the handle of the folder being synchronized. Specify the **PartialItem** flag in the **SendOptions** field (section 2.2.3.1.1.1.2) when using this ROP.

4. **RopSynchronizationUploadStateStreamBegin** ROP (section 2.2.3.2.2.1) – Upload the ICS state property **MetaTagIdsetGiven** (section 2.2.1.1.1) by using the synchronization download context.

5. **RopSynchronizationUploadStateStreamContinue** ROP (section 2.2.3.2.2.2) – Upload the ICS state property **MetaTagIdsetGiven** by using the synchronization download context.

6. **RopSynchronizationUploadStateStreamEnd** ROP (section 2.2.3.2.2.3) – Upload the ICS state property **MetaTagIdsetGiven** by using the synchronization download context.

7. **RopSynchronizationUploadStateStreamBegin** ROP – Upload the ICS state property **MetaTagCnsetSeen** (section 2.2.1.1.2) by using the synchronization download context.

8. **RopSynchronizationUploadStateStreamContinue** ROP – Upload the ICS state property **MetaTagCnsetSeen** by using the synchronization download context.

9. **RopSynchronizationUploadStateStreamEnd** ROP – Upload the ICS state property **MetaTagCnsetSeen** by using the synchronization download context.

10. **RopSynchronizationUploadStateStreamBegin** ROP – Upload the ICS state property **MetaTagCnsetSeenFAI** (section 2.2.1.1.3) by using the synchronization download context.

11. **RopSynchronizationUploadStateStreamContinue** ROP – Upload the ICS state property **MetaTagCnsetSeenFAI** by using the synchronization download context.

12. **RopSynchronizationUploadStateStreamEnd** ROP – Upload the ICS state property **MetaTagCnsetSeenFAI** by using the synchronization download context.

13. **RopSynchronizationUploadStateStreamBegin** ROP – Upload the ICS state property **MetaTagCnsetRead** (section 2.2.1.1.4) by using the synchronization download context.

14. **RopSynchronizationUploadStateStreamContinue** ROP – Upload the ICS state property **MetaTagCnsetRead** by using the synchronization download context.

15. **RopSynchronizationUploadStateStreamEnd** ROP – Upload the ICS state property **MetaTagCnsetRead** by using the synchronization download context.

16. **RopFastTransferSourceGetBuffer** ROP (section 2.2.3.2.2.3) – Receive the folder properties and updated ICS state by using the synchronization download context. These buffers contain partial items as appropriate.

17. **RopRelease** ROP ([MS-OXCROPS] section 2.2.15.3) – Release the synchronization download context.

18. **RopRelease** ROP – Release the folder.

19. **RopRelease** ROP – Release the message store.

4.4 Serialization of an IDSET Structure Example

To efficiently transfer large numbers of **Message ID** structures ([MS-OXCDATA] section 2.2.1.2) and **Folder ID** structures ([MS-OXCDATA] section 2.2.1.1) that identify changed or new messaging objects, the **Message ID** values and the **Folder ID** values are serialized into an **IDSET** structure for transfer across the wire. The following example shows how to format and serialize an **IDSET**. Because of the variability of the **GLOBSET** structure encoding commands that are used within the serialization of an **IDSET**, an **IDSET** can be encoded in many different ways. There is no single correct way to encode a **GLOBSET** as long as the **GLOBSET**, when decoded, contains the same set of **GLOBCNT** values, as specified in section 2.2.2.5. The following is just one way to encode an **IDSET**.

This example uses an **IDSET** with the four following **Message ID** values:

IDSET Structure

Message name	Value	REPLID	GLOBCNT
MessageID1	01 00 00 00 00 00 00 05	0001	000000000005
MessageID2	01 00 00 00 00 00 00 06	0001	000000000006
MessageID3	01 00 00 00 00 00 00 10	0001	000000000010
MessageID4	02 00 00 00 00 00 00 09	0002	000000000009

The **IDSET** has to be properly formatted for serializations. For more details about how to format an **IDSET**, see section 3.1.5.4.1.

The following diagram represents how the **IDSET** has to be arranged for serialization. The individual ID values have been arranged by **REPLID** and the **GLOBCNT** values have been reduced to a **GLOBSET** for each **REPLID**. Within the **GLOBSET**, the **GLOBCNT** values are placed into contiguous ranges.

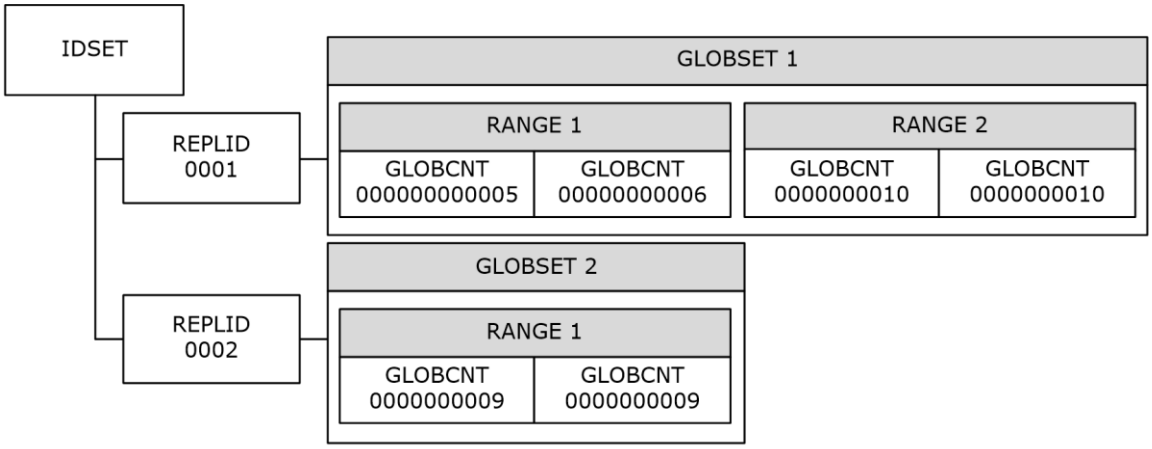


Figure 5: Arranging the IDSET structure for serialization

This example serializes the **IDSET** by using the **REPLID** format. For more details about the different serialization formats of an **IDSET**, see section 2.2.2.4.

For each **REPLID**/**GLOBSET** pair, the **REPLID** has to be added to the serialization buffer before the encoded **GLOBSET**. They have to be ordered based on the **REPLID** value where they are ordered from lowest to highest value.

The serialization buffer resembles the following:

Serialization Buffer
01 00 <encoded GLOBSET 1> 02 00 <encoded GLOBSET 2>

GLOBSET 1 contains four **GLOBCNT** values; two in each **GLOBCNT** range. The encoding has to be performed based on the same order in which they are arranged in **GLOBCNT** ranges: from lowest to highest value. The following table is a list of all the **GLOBCNT** values in the order in which they have to be encoded.

#	**GLOBCNT**
1	00 00 00 00 00 05
2	00 00 00 00 00 06
3	00 00 00 00 00 10
4	00 00 00 00 00 10

Because all values have the same five bytes in common, the **Push** command can be used to push the five common bytes onto the common byte stack.

Current Encoding Buffer
05 00 00 00 00 00

Low and high **GLOBCNT** values in all ranges have to be evaluated in pairs. Because value 1 is close to value 2, it is possible to continue to evaluate subsequent ranges of **GLOBCNT** values to see if the

Bitmask command can be used. However, values 3 and 4 are not close enough to value 1 to use the **Bitmask** command. Because only one **GLOBCNT** range is put into a **Bitmask** command, either the **Bitmask** command or the **Range** command could be used. Because they both occupy the same number of bytes in the encoded buffer, whether to use a **Bitmask** or **Range** command is an implementation decision. Both methods when decoded result in the same **GLOBCNT** range. In this example, the **Range** command is used with the values 0x05 and 0x06 following it.

Current Encoding Buffer
05 00 00 00 00 00 **52 05 06**

This results in encodings to generate **GLOBCNT** values 1 and 2 if decoded. For **GLOBCNT** value 3 and 4, because they both have five bytes in common that are already in the common byte stack, no **Pop** or **Push** command has to be used. Because values 3 and 4 are close in value (in this particular case, they are identical), the **Bitmask** command could be used. Because there are no more **GLOBCNT** ranges to encode, the **Bitmask** command only contains one range that takes 3 bytes of encoding. This is the same size a **Range** command would be to encode the same range. However, because the range is a singleton, it is more efficient to use the **Push** command to fill in the common byte stack. This generates two identical **GLOBCNT** values when decoded.

Current Encoding Buffer
05 00 00 00 00 00 52 05 06 **01 10**

This results in encodings in the encoding buffer to generate all **GLOBCNT** values in the **GLOBSET**. To complete the encoding, an **End** command has to be added. Before the **End** command can be added, any bytes on the common byte stack have to be removed. Because all bytes on the common byte stack were pushed with a single **Push** command, only one **Pop** command is required to remove them.

Current Encoding Buffer
05 00 00 00 00 00 52 05 06 01 10 **50**

The **End** command can now be added.

Current Encoding Buffer
05 00 00 00 00 00 52 05 06 01 10 50 **00**

The GLOBSET 1 encoding can be added to the serialization buffer to produce the following:

Serialization Buffer
01 00 **05 00 00 00 00 00 52 05 06 01 10 50 00** 02 00 <encoded GLOBSET 2>

The last step is to encode GLOBSET 2. GLOBSET 2 contains two **GLOBCNT** values. The following table is a list of all the **GLOBCNT** values in the order in which they have to be encoded.

#	GLOBCNT
1	00 00 00 00 00 09
2	00 00 00 00 00 09

Because both **GLOBCNT** values 1 and 2 are identical, the **Push** command can be used, followed by the full 6 bytes to add to the common byte stack. Because this fills the common array, it generates two identical **GLOBCNT** values when decoded, producing a singleton **GLOBCNT** range.

Current Encoding Buffer
06 00 00 00 00 00 09

Encodings in the encoding buffer now exist to generate all **GLOBCNT** values in the **GLOBSET**. To complete the encoding, an **End** command has to be added.

Current Encoding Buffer
06 00 00 00 00 00 09 **00**

The GLOBSET 2 encoding can be added to the serialization buffer to produce the following:

Serialization Buffer
01 00 05 00 00 00 00 00 52 05 06 01 10 50 00 02 00 **06 00 00 00 00 00 09 00**

This completes the serialization of the **IDSET**.

4.5 FastTransfer Stream Produced by a Content Synchronization Download Example

The following example shows the sample output of a FastTransfer stream that is downloaded to a client during a content synchronization operation. The download operation was configured by using the **RopSynchronizationConfigure** ROP (section 2.2.3.2.1.1) with the following fields specified in the request buffer:

Request buffer field	Value
SynchronizationType	**Contents**
SendOptions	**Unicode**, **RecoverMode**, **ForceUnicode**, and **PartialItem** flags
SynchronizationFlags	**Unicode**, **ReadState**, **FAI**, **Normal**, **NoForeignIdentifiers**, **BestBody**, and **Progress** flags
RestrictionDataSize	0
RestrictionData	< missing >
SynchronizationExtraFlags	**Eid**, **CN**, **OrderByDeliveryTime** flags

The FastTransfer stream contains the full message change for one message, message deletions, message read state changes, and the final ICS state. The following list shows the structure of the data included in this FastTransfer stream. The list shows the markers that occur in this stream in the order of their appearance. The nesting structure shows the logical relationship of the data delimited by the markers.

```
IncrSyncProgressMode
    IncrSyncProgressPerMsg
    IncrSyncChg
        IncrSyncMessage
            StartRecip
            EndToRecip
            NewAttach
                StartEmbed
                    StartRecip
                    EndToRecip
                EndEmbed
            EndAttach
    IncrSyncDel
```

```
        IncrSyncRead
        IncrSyncStateBegin
        IncrSyncStateEnd
    IncrSyncEnd
```

In the following table, certain property tags are identified as special property tags, which means that they contain 0000 for a **property ID**, and the meaning of the property is determined by the context of the property in the stream.

Bytes on the wire	Value/description
0B 00 74 40	**marker** **IncrSyncProgressMode** marker (section 2.2.4.1.4) (4074000B [Bool])
02 01 00 00	propDef ProgressInformation (special) (00000102 [Binary])
20 00 00 00	length 32 (0x20)
26 00 00 00- 32 54 76 98 BE BA BE BA- BE BA BE BA EF CD AB 00- 00 00 00 00 EF CD AB 90- 78 56 34 12	varSizeValue
0B 00 75 40	**marker** **IncrSyncProgressPerMsg** marker (section 2.2.4.1.4) (4075000B [Bool])
03 00 00 00	propDef MessageSize (special) (00000003 [Int32])
38 00 00 00	fixedSizeValue [Int32] 56
0B 00 00 00	propDef IsAssociated (special) (0000000B [Bool])
00 00	fixedSizeValue [Bool] **FALSE**
03 00 12 40	**marker** **IncrSyncChg marker (section 2.2.4.1.4) (40120003 [Int32])**
02 01 E0 65	propDef **PidTagSourceKey** property (section 2.2.1.2.5) (65E00102 [Binary])
16 00 00 00	length 22 (0x16)
19 D7 FB 0F- 06 16 A1 41 BF F6 91 C7- 63 DA A8 66	varSizeValue ....A ....c..f

Bytes on the wire	Value/description
00 00 00 78-2E 21	...x.!
40 00 08 30	propDef **PidTagLastModificationTime** property ([MS-OXPROPS] section 2.755) (30080040 [SysTime])
FC 65 69 CF-C0 84 C8 01	fixedSizeValue [SysTime] 2008-03-13T04:15:02.8437500
02 01 E2 65	propDef **PidTagChangeKey** property (section 2.2.1.2.7) (65E20102 [Binary])
16 00 00 00	length 22 (0x16)
19 D7 FB 0F-06 16 A1 41 BF F6 91 C7-63 DA A8 66 00 00 00 78-4D 1C	varSizeValue ....A ....c..f ...xM.
02 01 E3 65	propDef **PidTagPredecessorChangeList** property (section 2.2.1.2.8) (65E30102 [Binary])
17 00 00 00	length 23 (0x17)
16 19 D7 FB-0F 06 16 A1 41 BF F6 91-C7 63 DA A8 66 00 00 00-78 4D 1C	varSizeValue .... A....c.. f...xM.
0B 00 AA 67	propDef **PidTagAssociated** property (section 2.2.1.5) (67AA000B [Bool])
00 00	fixedSizeValue [Bool] **FALSE**
14 00 4A 67	propDef **PidTagMid** property (section 2.2.1.2.1) (674A0014 [Int64])
01 00 00 00-00 78 2E 21	fixedSizeValue [Int64] 2390980393575645185
14 00 A4 67	propDef **PidTagChangeNumber** property (section 2.2.1.2.3) (67A40014 [Int64])
01 00 00 00-00 78 4D 1C	fixedSizeValue [Int64] 2039418147664035841
03 00 15 40	**marker** **IncrSyncMessage** marker (section 2.2.4.1.4) (40150003 [Int32])
0B 00 02 00	propDef

Bytes on the wire	Value/description
	PidTagAlternateRecipientAllowed property ([MS-OXPROPS] section 2.568) (0002000B [Bool])
01 00	fixedSizeValue [Bool] **TRUE**
03 00 17 00	propDef **PidTagImportance** property ([MS-OXCMSG] section 2.2.1.11) (00170003 [Int32])
01 00 00 00	fixedSizeValue [Int32] 1
1F 00 1A 00	propDef **PidTagMessageClass** property ([MS-OXCMSG] section 2.2.1.3) (001A001F [Unicode])
12 00 00 00	length 18 (0x12)
49 00 50 00-4D 00 2E 00 4E 00 6F 00-74 00 65 00 00 00	varSizeValue I.P.M... N.o.t.e. ..
0B 00 23 00	propDef **PidTagOriginatorDeliveryReportRequested** property ([MS-OXOMSG] section 2.2.1.20) (0023000B [Bool])
00 00	fixedSizeValue [Bool] **FALSE**
03 00 26 00	propDef **PidTagPriority** property ([MS-OXCMSG] section 2.2.1.12) (00260003 [Int32])
00 00 00 00	fixedSizeValue [Int32] 0
0B 00 29 00	propDef **PidTagReadReceiptRequested** property ([MS-OXOMSG] section 2.2.1.29) (0029000B [Bool])
00 00	fixedSizeValue [Bool] **FALSE**
03 00 36 00	propDef **PidTagSensitivity** property ([MS-OXCMSG] section 2.2.1.13) (00360003 [Int32])
00 00 00 00	fixedSizeValue [Int32] 0
1F 00 37 00	propDef **PidTagSubject** property ([MS-OXPROPS] section 2.1023) (0037001F [Unicode])
26 00 00 00	length 38 (0x26)
54 00 65 00-73 00 74 00	varSizeValue

Bytes on the wire	Value/description
20 00 77 00- 69 00 74 00 68 00 20 00- 65 00 6D 00 62 00 65 00- 64 00 64 00 65 00 64 00- 00 00	T.e.s.t. .w.i.t. h..e.m. b.e.d.d. e.d...
... value truncated...	
40 00 39 00	propDef **PidTagClientSubmitTime** property ([MS-OXOMSG] section 2.2.3.11) (00390040 [SysTime])
80 BA A7 B7- BC 84 C8 01	fixedSizeValue [SysTime] 2008-03-13T03:45:45.0000000
02 01 3B 00	propDef **PidTagSentRepresentingSearchKey** property ([MS-OXOMSG] section 2.2.1.58) (003B0102 [Binary])
60 00 00 00	length 96 (0x60)
45 58 3A 2F- 4F 3D 46 49 52 53 54 20- 4F 52 47 41 4E 49 5A 41- 54 49 4F 4E 2F 4F 55 3D- 45 58 43 48 41 4E 47 45- 20 41 44 4D	varSizeValue EX:/O=FI RST ORGA NIZATION /OU=EXCH ANGE ADM
... value truncated...	
1F 00 3D 00	propDef **PidTagSubjectPrefix** property ([MS-OXCMSG] section 2.2.1.9) (003D001F [Unicode])
02 00 00 00	length 2 (0x2)
00 00	varSizeValue ..
02 01 3F 00	propDef **PidTagReceivedByEntryId** property ([MS-OXOMSG] section 2.2.1.38) (003F0102 [Binary])
79 00 00 00	length 121 (0x79)
00 00 00 00- DC A7 40 C8 C0 42 10 1A- B4 B9 08 00 2B 2F E1 82-	varSizeValue .....@. .B..... +/.....

Bytes on the wire	Value/description
01 00 00 00 00 00 00 00-2F 4F 3D 46 49 52 53 54-20 4F 52 47	..../O=F IRST ORG
... value truncated...	
1F 00 40 00	propDef **PidTagReceivedByName** property ([MS-OXOMSG] section 2.2.1.39) (0040001F [Unicode])
06 00 00 00	length 6 (0x6)
74 00 31 00-00 00	varSizeValue t.1...
02 01 41 00	propDef **PidTagSentRepresentingEntryId** property ([MS-OXOMSG] section 2.2.1.56) (00410102 [Binary])
79 00 00 00	length 121 (0x79)
00 00 00 00-DC A7 40 C8 C0 42 10 1A-B4 B9 08 00 2B 2F E1 82-01 00 00 00 00 00 00 00-2F 4F 3D 46 49 52 53 54-20 4F 52 47	varSizeValue ......@. .B...... +/..... ..../O=F IRST ORG
... value truncated...	
1F 00 42 00	propDef **PidTagSentRepresentingName** property ([MS-OXOMSG] section 2.2.1.57) (0042001F [Unicode])
06 00 00 00	length 6 (0x6)
74 00 31 00-00 00	varSizeValue t.1...
02 01 43 00	propDef **PidTagReceivedRepresentingEntryId** property ([MS-OXOMSG] section 2.2.1.56) (00430102 [Binary])
79 00 00 00	length 121 (0x79)
00 00 00 00-DC A7 40 C8 C0 42 10 1A-	varSizeValue ......@.

Bytes on the wire	Value/description
B4 B9 08 00 2B 2F E1 82-01 00 00 00 00 00 00 00-2F 4F 3D 46 49 52 53 54-20 4F 52 47	.B..... +/..... ..../O=F IRST ORG
... value truncated...	
1F 00 44 00	propDef **PidTagReceivedRepresentingName** property ([MS-OXOMSG] section 2.2.1.57) (0044001F [Unicode])
06 00 00 00	length 6 (0x6)
74 00 31 00-00 00	varSizeValue t.1...
02 01 51 00	propDef **PidTagReceivedBySearchKey** property ([MS-OXOMSG] section 2.2.1.40) (00510102 [Binary])
60 00 00 00	length 96 (0x60)
45 58 3A 2F-4F 3D 46 49 52 53 54 20-4F 52 47 41 4E 49 5A 41-54 49 4F 4E 2F 4F 55 3D-45 58 43 48 41 4E 47 45-20 41 44 4D	varSizeValue EX:/O=FI RST ORGA NIZATION /OU=EXCH ANGE ADM
... value truncated...	
02 01 52 00	propDef **PidTagReceivedRepresentingSearchKey** property ([MS-OXOMSG] section 2.2.1.27) (00520102 [Binary])
60 00 00 00	length 96 (0x60)
45 58 3A 2F-4F 3D 46 49 52 53 54 20-4F 52 47 41 4E 49 5A 41-54 49 4F 4E 2F 4F 55 3D-45 58 43 48 41 4E 47 45-20 41 44 4D	varSizeValue EX:/O=FI RST ORGA NIZATION /OU=EXCH ANGE ADM

Bytes on the wire	Value/description
... value truncated...	
1F 00 64 00	propDef **PidTagSentRepresentingAddressType** property ([MS-OXOMSG] section 2.2.1.54) (0064001F [Unicode])
06 00 00 00	length 6 (0x6)
45 00 58 00-00 00	varSizeValue E.X...
1F 00 65 00	propDef **PidTagSentRepresentingEmailAddress** ([MS-OXOMSG] section 2.2.1.55) (0065001F [Unicode])
BA 00 00 00	length 186 (0xBA)
2F 00 4F 00-3D 00 46 00 49 00 52 00-53 00 54 00 20 00 4F 00-52 00 47 00 41 00 4E 00-49 00 5A 00 41 00 54 00-49 00 4F 00	varSizeValue /.O.=.F. I.R.S.T. .O.R.G. A.N.I.Z. A.T.I.O.
... value truncated...	
1F 00 70 00	propDef **PidTagConversationTopic** property ([MS-OXOMSG] section 2.2.1.5) (0070001F [Unicode])
26 00 00 00	length 38 (0x26)
54 00 65 00-73 00 74 00 20 00 77 00-69 00 74 00 68 00 20 00-65 00 6D 00 62 00 65 00-64 00 64 00 65 00 64 00-00 00	varSizeValue T.e.s.t. .w.i.t. h..e.m. b.e.d.d. e.d...
... value truncated...	
02 01 71 00	propDef **PidTagConversationIndex** property ([MS-OXOMSG] section 2.2.1.3) (00710102 [Binary])
16 00 00 00	length 22 (0x16)

Bytes on the wire	Value/description
01 C8 84 BC-B6 CB 8A CC 1E B8 32 77-43 2B A1 C6 83 9A 4A F4-BC 14	varSizeValue ....2wC+.. ..J...
1F 00 75 00	propDef **PidTagReceivedByAddressType** ([MS-OXOMSG] section 2.2.1.36) (0075001F [Unicode])
06 00 00 00	length 6 (0x6)
45 00 58 00-00 00	varSizeValue E.X...
1F 00 76 00	propDef **PidTagReceivedByEmailAddress** property ([MS-OXOMSG] section 2.2.1.37) (0076001F [Unicode])
BA 00 00 00	length 186 (0xBA)
2F 00 4F 00-3D 00 46 00 49 00 52 00-53 00 54 00 20 00 4F 00-52 00 47 00 41 00 4E 00-49 00 5A 00 41 00 54 00-49 00 4F 00	varSizeValue /.O.=.F. I.R.S.T. .O.R.G. A.N.I.Z. A.T.I.O.
... value truncated...	
1F 00 77 00	propDef **PidTagReceivedRepresentingAddressType** property ([MS-OXOMSG] section 2.2.1.23) (0077001F [Unicode])
06 00 00 00	length 6 (0x6)
45 00 58 00-00 00	varSizeValue E.X...
1F 00 78 00	propDef **PidTagReceivedRepresentingEmailAddress** property ([MS-OXOMSG] section 2.2.1.24) (0078001F [Unicode])
BA 00 00 00	length 186 (0xBA)
2F 00 4F 00-3D 00 46 00 49 00 52 00-	varSizeValue /.O.=.F. I.R.S.T.

Bytes on the wire	Value/description
53 00 54 00 20 00 4F 00- 52 00 47 00 41 00 4E 00- 49 00 5A 00 41 00 54 00- 49 00 4F 00	.O.R.G. A.N.I.Z. A.T.I.O.
... value truncated...	
1F 00 7D 00	propDef **PidTagTransportMessageHeaders** property ([MS-OXOMSG] section 2.2.1.61) (007D001F [Unicode])
E8 06 00 00	length 1768 (0x6E8)
52 00 65 00- 63 00 65 00 69 00 76 00- 65 00 64 00 3A 00 20 00- 66 00 72 00 6F 00 6D 00- 20 00 45 00 58 00 43 00- 48 00 2D 00	varSizeValue R.e.c.e. i.v.e.d. :..f.r. o.m..E. X.C.H.-.
... value truncated...	
02 01 7F 00	propDef **PidTagTnefCorrelationKey** property ([MS-OXPROPS] section 2.1037) (007F0102 [Binary])
56 00 00 00	length 86 (0x56)
3C 31 39 44- 37 46 42 30 46 30 36 31- 36 41 31 34 31 42 46 46- 36 39 31 43 37 36 33 44- 41 41 38 36 36 37 38 34- 34 42 37 40	varSizeValue <19D7FB0 F0616A14 1BFF691C 763DAA86 67844B7@
... value truncated...	
02 01 19 0C	propDef **PidTagSenderEntryId** property ([MS-OXOMSG] section 2.2.1.50) (0C190102 [Binary])
79 00 00 00	length 121 (0x79)
00 00 00 00- DC A7 40 C8	varSizeValue

Bytes on the wire	Value/description
C0 42 10 1A- B4 B9 08 00 2B 2F E1 82- 01 00 00 00 00 00 00 00- 2F 4F 3D 46 49 52 53 54- 20 4F 52 47	......@. .B...... +/..... ..../O=F IRST ORG
... value truncated...	
1F 00 1A 0C	propDef **PidTagSenderName** property ([MS-OXOMSG] section 2.2.1.51) (0C1A001F [Unicode])
06 00 00 00	length 6 (0x6)
74 00 31 00- 00 00	varSizeValue t.1...
02 01 1D 0C	propDef **PidTagSenderSearchKey** property ([MS-OXOMSG] section 2.2.1.52) (0C1D0102 [Binary])
60 00 00 00	Length 96 (0x60)
45 58 3A 2F- 4F 3D 46 49 52 53 54 20- 4F 52 47 41 4E 49 5A 41- 54 49 4F 4E 2F 4F 55 3D- 45 58 43 48 41 4E 47 45- 20 41 44 4D	varSizeValue EX:/O=FI RST ORGA NIZATION /OU=EXCH ANGE ADM
... value truncated...	
1F 00 1E 0C	propDef **PidTagSenderAddressType** property ([MS-OXOMSG] section 2.2.1.48) (0C1E001F [Unicode])
06 00 00 00	length 6 (0x6)
45 00 58 00- 00 00	varSizeValue E.X...
1F 00 1F 0C	propDef **PidTagSenderEmailAddress** property ([MS-OXOMSG] section 2.2.1.49) (0C1F001F [Unicode])
BA 00 00 00	length 186 (0xBA)
2F 00 4F 00- 3D 00 46 00 49 00 52 00-	varSizeValue /.O.=.F.

Bytes on the wire	Value/description
53 00 54 00 20 00 4F 00- 52 00 47 00 41 00 4E 00- 49 00 5A 00 41 00 54 00- 49 00 4F 00	I.R.S.T. .O.R.G. A.N.I.Z. A.T.I.O.
... value truncated...	
03 00 D3 83- 03 20 06 00 00 00 00 00- C0 00 00 00 00 00 00 46- 00 2A 81 00 00	propDef **PidLidTaskAcceptanceState** property ([MS-OXOTASK] section 2.2.2.2.30) (0x812A [PSETID_Task]) [Int32]
00 00 00 00	fixedSizeValue [Int32] 0
0B 00 D2 83- 03 20 06 00 00 00 00 00- C0 00 00 00 00 00 00 46- 00 2C 81 00 00	propDef **PidLidTaskFFixOffline** property ([MS-OXOTASK] section 2.2.2.2.31) (0x812C [PSETID_Task]) [Bool]
00 00	fixedSizeValue [Bool] **FALSE**
0B 00 D1 83- 03 20 06 00 00 00 00 00- C0 00 00 00 00 00 00 46- 00 24 81 00 00	propDef **PidLidTaskNoCompute** property ([MS-OXOTASK] section 2.2.2.2.35) (0x8124 [PSETID_Task]) [Bool]
00 00	fixedSizeValue [Bool] **FALSE**
40 00 06 0E	propDef **PidTagMessageDeliveryTime** property ([MS-OXOMSG] section 2.2.3.9) (0E060040 [SysTime])
80 E7 D8 B8- BC 84 C8 01	fixedSizeValue [SysTime] 2008-03-13T03:45:47.0000000
03 00 07 0E	propDef **PidTagMessageFlags** property ([MS-OXCMSG] section 2.2.1.6) (0E070003 [Int32])
31 00 00 00	fixedSizeValue [Int32] 49

Bytes on the wire	Value/description
03 00 CE 83-03 20 06 00 00 00 00 00-C0 00 00 00 00 00 00 46-00 29 81 00 00	propDef **PidLidTaskOwnership** property ([MS-OXOTASK] section 2.2.2.2.29) (0x8129 [PSETID_Task]) [Int32]
00 00 00 00	fixedSizeValue [Int32] 0
03 00 17 0E	propDef **PidTagMessageStatus** property ([MS-OXCMSG] section 2.2.1.8) (0E170003 [Int32])
00 00 00 00	fixedSizeValue [Int32] 0
03 00 D0 83-03 20 06 00 00 00 00 00-C0 00 00 00 00 00 00 46-00 11 81 00 00	propDef **PidLidTaskEstimatedEffort** property ([MS-OXOTASK] section 2.2.2.2.12) (0x8111 [PSETID_Task]) [Int32]
00 00 00 00	fixedSizeValue [Int32] 0
1F 00 1D 0E	propDef **PidTagNormalizedSubject** property ([MS-OXCMSG] section 2.2.1.10) (0E1D001F [Unicode])
26 00 00 00	length 38 (0x26)
54 00 65 00-73 00 74 00 20 00 77 00-69 00 74 00 68 00 20 00-65 00 6D 00 62 00 65 00-64 00 64 00 65 00 64 00-00 00	varSizeValue T.e.s.t. .w.i.t. h..e.m. b.e.d.d. e.d...
... value truncated...	
0B 00 1F 0E	propDef **PidTagRtfInSync** property ([MS-OXCMSG] section 2.2.1.56.5) (0E1F000B [Bool])
01 00	fixedSizeValue [Bool] **TRUE**
03 00 23 0E	propDef Unspecified property (0E230003 [Int32])

Bytes on the wire	Value/description
26 00 00 00	fixedSizeValue [Int32] 38
03 00 79 0E	propDef **PidTagTrustSender** property ([MS-OXPROPS] section 2.1041) (0E790003 [Int32])
01 00 00 00	fixedSizeValue [Int32] 1
03 00 CF 83-03 20 06 00 00 00 00 00-C0 00 00 00 00 00 00 46-00 10 81 00 00	propDef **PidLidTaskActualEffort** property ([MS-OXOTASK] section 2.2.2.2.11) (0x8110 [PSETID_Task]) [Int32]
00 00 00 00	fixedSizeValue [Int32] 0
03 00 F7 0F	propDef **PidTagAccessLevel** property ([MS-OXCPRPT] section 2.2.1.2) (0FF70003 [Int32])
00 00 00 00	fixedSizeValue [Int32] 0
1F 00 CD 83-03 20 06 00 00 00 00 00-C0 00 00 00 00 00 00 46-00 21 81 00 00	propDef **PidLidTaskAssigner** property ([MS-OXOTASK] section 2.2.2.2.24) (0x8121 [PSETID_Task]) [Unicode]
02 00 00 00	length 2 (0x2)
00 00	varSizeValue ..
03 00 CC 83-03 20 06 00 00 00 00 00-C0 00 00 00 00 00 00 46-00 23 81 00 00	propDef **PidLidTaskOrdinal** property ([MS-OXOTASK] section 2.2.2.2.26) (0x8123 [PSETID_Task]) [Int32]
FF FF FF 7F	fixedSizeValue [Int32] 2147483647
1F 00 35 10	propDef **PidTagInternetMessageId** property ([MS-OXOMSG] section 2.2.1.12) (1035001F [Unicode])
AC 00 00 00	length

Bytes on the wire	Value/description
	172 (0xAC)
3C 00 31 00-39 00 44 00 37 00 46 00-42 00 30 00 46 00 30 00-36 00 31 00 36 00 41 00-31 00 34 00 31 00 42 00-46 00 46 00	varSizeValue <.1.9.D. 7.F.B.0. F.0.6.1. 6.A.1.4. 1.B.F.F.
... value truncated...	
03 00 80 10	propDef **PidTagIconIndex** property ([MS-OXOMSG] section 2.2.1.10) (10800003 [Int32])
FF FF FF FF	fixedSizeValue [Int32] -1
40 00 07 30	propDef **PidTagCreationTime** property ([MS-OXCMSG] section 2.2.2.3) (30070040 [SysTime])
A2 DA EF B9-BC 84 C8 01	fixedSizeValue [SysTime] 2008-03-13T03:45:48.8281250
40 00 08 30	propDef **PidTagLastModificationTime** property (30080040 [SysTime])
FC 65 69 CF-C0 84 C8 01	fixedSizeValue [SysTime] 2008-03-13T04:15:02.8437500
02 01 0B 30	propDef **PidTagSearchKey** property ([MS-OXCPRPT] section 2.2.1.9) (300B0102 [Binary])
10 00 00 00	length 16 (0x10)
6B 3B AA B8-C7 83 78 4E 80 8E F2 DE-04 82 C8 EB	varSizeValue k;....xN .....
0B 00 40 3A	propDef **PidTagSendRichInfo** property ([MS-OXOMSG] section 2.2.1.47) (3A40000B [Bool])
01 00	fixedSizeValue [Bool] **TRUE**
03 00 DE 3F	propDef **PidTagInternetCodepage** property ([MS-OXCMSG] section 2.2.1.56.6) (3FDE0003 [Int32])
9F 4E 00 00	fixedSizeValue [Int32] 20127
03 00 F1 3F	propDef

Bytes on the wire	Value/description
	PidTagMessageLocaleId property ([MS-OXCMSG] section 2.2.1.5) (3FF10003 [Int32])
09 04 00 00	fixedSizeValue [Int32] 1033
03 00 FD 3F	propDef **PidTagMessageCodepage** property ([MS-OXCMSG] section 2.2.1.4) (3FFD0003 [Int32])
E3 04 00 00	fixedSizeValue [Int32] 1251
03 00 19 40	propDef Unspecified property (40190003 [Int32])
00 00 00 00	fixedSizeValue [Int32] 0
03 00 1A 40	propDef **PidTagSentRepresentingFlags** property ([MS-OXPROPS] section 2.1006) (401A0003 [Int32])
00 00 00 00	fixedSizeValue [Int32] 0
03 00 1B 40	propDef Unspecified property (401B0003 [Int32])
00 00 00 00	fixedSizeValue [Int32] 0
03 00 1C 40	propDef Unspecified property (401C0003 [Int32])
00 00 00 00	fixedSizeValue [Int32] 0
03 00 76 40	propDef **PidTagContentFilterSpamConfidenceLevel** property ([MS-OXCSPAM] section 2.2.1.3) (40760003 [Int32])
FF FF FF FF	fixedSizeValue [Int32] -1
03 00 02 59	propDef **PidTagInternetMailOverrideFormat** property ([MS-OXOMSG] section 2.2.1.11) (59020003 [Int32])
00 00 16 00	fixedSizeValue [Int32] 1441792
03 00 09 59	propDef **PidTagMessageEditorFormat** property ([MS-OXPROPS] section 2.781) (59090003 [Int32])
02 00 00 00	fixedSizeValue [Int32] 2
03 00 C6 65	propDef

Bytes on the wire	Value/description
	Unspecified property (65C60003 [Int32])
02 00 00 00	fixedSizeValue [Int32] 2
1F 00 D4 83-03 20 06 00 00 00 00 00-C0 00 00 00 00 00 00 46-00 27 81 00 00	propDef **PidLidTaskRole** property ([MS-OXPROPS] section 2.332) (0x8127 [PSETID_Task]) [Unicode]
02 00 00 00	length 2 (0x2)
00 00	varSizeValue ..
0B 00 D5 83-03 20 06 00 00 00 00 00-C0 00 00 00 00 00 00 46-00 03 81 00 00	propDef **PidLidTeamTask** property ([MS-OXOTASK] section 2.2.2.2.36) (0x8103 [PSETID_Task]) [Bool]
00 00	fixedSizeValue [Bool] **FALSE**
0B 00 D6 83-03 20 06 00 00 00 00 00-C0 00 00 00 00 00 00 46-00 26 81 00 00	propDef **PidLidTaskFRecurring** property ([MS-OXOTASK] section 2.2.2.2.28) (0x8126 [PSETID_Task]) [Bool]
00 00	fixedSizeValue [Bool] **FALSE**
03 00 00 80-08 20 06 00 00 00 00 00-C0 00 00 00 00 00 00 46-00 52 85 00 00	propDef **PidLidCurrentVersion** property ([MS-OXPROPS] section 2.88) (0x8552 [PSETID_Common]) [Int32]
04 ED 01 00	fixedSizeValue [Int32] 126212
1F 00 01 80-08 20 06 00 00 00 00 00-C0 00 00 00	propDef **PidLidCurrentVersionName** property ([MS-OXPROPS] section 2.89) (0x8554 [PSETID_Common]) [Unicode]

Bytes on the wire	Value/description
00 00 00 46- 00 54 85 00 00	
0A 00 00 00	length 10 (0xA)
31 00 32 00- 2E 00 30 00 00 00	varSizeValue 1.2...0. ..
03 00 02 80- 08 20 06 00 00 00 00 00- C0 00 00 00 00 00 00 46- 00 10 85 00 00	propDef **PidLidSideEffects** property ([MS-OXCMSG] section 2.2.1.16) (0x8510 [PSETID_Common]) [Int32]
00 00 00 00	fixedSizeValue [Int32] 0
0B 00 08 80- 08 20 06 00 00 00 00 00- C0 00 00 00 00 00 00 46- 00 03 85 00 00	propDef **PidLidReminderSet** property ([MS-OXORMDR] section 2.2.1.1) (0x8503 [PSETID_Common]) [Bool]
00 00	fixedSizeValue [Bool] **FALSE**
1F 10 0C 80- 29 03 02 00 00 00 00 00- C0 00 00 00 00 00 00 46- 01 4B 00 65 00 79 00 77- 00 6F 00 72 00 64 00 73- 00 00 00	propDef **PidNameKeywords** property ([MS-OXCMSG] section 2.2.1.17) (Keywords [PS_PUBLIC_STRINGS]) [MultiValueUnicode]
02 00 00 00	length 2 (0x2)
1C 00 00 00	length 28 (0x1C)
42 00 6C 00- 75 00 65 00 20 00 43 00- 61 00 74 00 65 00 67 00- 6F 00 72 00	varSizeValue B.l.u.e. .C.a.t. e.g.o.r. y...

Bytes on the wire	Value/description
79 00 00 00	
20 00 00 00	length 32 (0x20)
59 00 65 00-6C 00 6C 00 6F 00 77 00-20 00 43 00 61 00 74 00-65 00 67 00 6F 00 72 00-79 00 00 00	varSizeValue Y.e.l.l. o.w..C. a.t.e.g. o.r.y...
0B 00 4D 81-08 20 06 00 00 00 00 00-C0 00 00 00 00 00 00 46-00 0E 85 00 00	propDef **PidLidAgingDontAgeMe** property ([MS-OXPROPS] section 2.4) (0x850E [PSETID_Common]) [Bool]
00 00	fixedSizeValue [Bool] **FALSE**
03 00 84 81-08 20 06 00 00 00 00 00-C0 00 00 00 00 00 00 46-00 18 85 00 00	propDef **PidLidTaskMode** property ([MS-OXOTASK] section 2.2.2.2.1) (0x8518 [PSETID_Common]) [Int32]
00 00 00 00	fixedSizeValue [Int32] 0
0B 00 4B 82-08 20 06 00 00 00 00 00-C0 00 00 00 00 00 00 46-00 06 85 00 00	propDef **PidLidPrivate** property ([MS-OXCMSG] section 2.2.1.15) (0x8506 [PSETID_Common]) [Bool]
00 00	fixedSizeValue [Bool] **FALSE**
1F 00 4D 82-08 20 06 00 00 00 00 00-C0 00 00 00 00 00 00 46-00 80 85 00 00	propDef **PidLidInternetAccountName** property ([MS-OXOMSG] section 2.2.1.62) (0x8580 [PSETID_Common]) [Unicode]
26 00 00 00	length

Bytes on the wire	Value/description
	38 (0x26)
4D 00 69 00-63 00 72 00 6F 00 73 00-6F 00 66 00 74 00 20 00-45 00 78 00 63 00 68 00-61 00 6E 00 67 00 65 00-00 00	varSizeValue M.i.c.r. o.s.o.f. t..E.x. c.h.a.n. g.e...
... value truncated...	
1F 00 4E 82-08 20 06 00 00 00 00 00-C0 00 00 00 00 00 00 46-00 81 85 00 00	propDef **PidLidInternetAccountStamp** property ([MS-OXOMSG] section 2.2.1.63) (0x8581 [PSETID_Common]) [Unicode]
E4 00 00 00	length 228 (0xE4)
30 00 30 00-30 00 30 00 30 00 30 00-30 00 32 00 01 00 45 00-58 00 43 00 48 00 2D 00-43 00 4C 00 49 00 2D 00-31 00 38 00	varSizeValue 0.0.0.0. 0.0.0.2. ..E.X.C. H.-.C.L. I.-.1.8.
... value truncated...	
0B 00 4F 82-08 20 06 00 00 00 00 00-C0 00 00 00 00 00 00 46-00 82 85 00 00	propDef **PidLidUseTnef** property ([MS-OXOMSG] section 2.2.1.66) (0x8582 [PSETID_Common]) [Bool]
00 00	fixedSizeValue [Bool] **FALSE**
03 00 A8 83-08 20 06 00 00 00 00 00-C0 00 00 00 00 00 00 46-00 01 85 00	propDef **PidLidReminderDelta** property ([MS-OXORMDR] section 2.2.1.3) (0x8501 [PSETID_Common]) [Int32]

Bytes on the wire	Value/description
00	
00 00 00 00	fixedSizeValue [Int32] 0
03 00 AD 83- 03 20 06 00 00 00 00 00- C0 00 00 00 00 00 00 46- 00 01 81 00 00	propDef **PidLidTaskStatus** property ([MS-OXOTASK] section 2.2.2.2.2) (0x8101 [PSETID_Task]) [Int32]
00 00 00 00	fixedSizeValue [Int32] 0
05 00 AE 83- 03 20 06 00 00 00 00 00- C0 00 00 00 00 00 00 46- 00 02 81 00 00	propDef **PidLidPercentComplete** property ([MS-OXOTASK] section 2.2.2.2.3) (0x8102 [PSETID_Task]) [Double]
00 00 00 00- 00 00 00 00	fixedSizeValue [Double] 0
0B 00 B0 83- 03 20 06 00 00 00 00 00- C0 00 00 00 00 00 00 46- 00 1C 81 00 00	propDef **PidLidTaskComplete** property ([MS-OXOTASK] section 2.2.2.2.20) (0x811C [PSETID_Task]) [Bool]
00 00	fixedSizeValue [Bool] **FALSE**
03 00 CA 83- 03 20 06 00 00 00 00 00- C0 00 00 00 00 00 00 46- 00 13 81 00 00	propDef **PidLidTaskState** property ([MS-OXOTASK] section 2.2.2.2.14) (0x8113 [PSETID_Task]) [Int32]
01 00 00 00	fixedSizeValue [Int32] 1
03 00 CB 83- 03 20 06 00 00 00 00 00- C0 00 00 00 00 00 00 46- 00 12 81 00 00	propDef **PidLidTaskVersion** property ([MS-OXOTASK] section 2.2.2.2.13) (0x8112 [PSETID_Task]) [Int32]

Bytes on the wire	Value/description
01 00 00 00	fixedSizeValue [Int32] 1
02 01 13 10	propDef **PidTagBodyHtml** property ([MS-OXCMSG] section 2.2.1.56.3) (10130102 [Binary])
58 06 00 00	length 1624 (0x658)
3C 68 74 6D-6C 20 78 6D 6C 6E 73 3A-76 3D 22 75 72 6E 3A 73-63 68 65 6D 61 73 2D 6D-69 63 72 6F 73 6F 66 74-2D 63 6F 6D	varSizeValue <html xm lns:v="u rn:schem as-micro soft-com
... value truncated...	
03 00 16 40	propDef **MetaTagFXDelProp** property (section 2.2.4.1.5.1) (40160003 [Int32])
0D 00 12 0E	fixedSizeValue **PidTagMessageRecipients** property ([MS-OXPROPS] section 2.786) (0E12000D [Object])
03 00 03 40	**marker** **StartRecip** marker (section 2.2.4.1.4) (40030003 [Int32])
03 00 00 30	propDef **PidTagRowid** property ([MS-OXPROPS] section 2.930) (30000003 [Int32])
00 00 00 00	fixedSizeValue [Int32] 0
1F 00 02 30	propDef **PidTagAddressType** property ([MS-OXCMAIL] section 2.1.3.1.9) (3002001F [Unicode])
06 00 00 00	length 6 (0x6)
45 00 58 00-00 00	varSizeValue E.X...
1F 00 03 30	propDef **PidTagEmailAddress** property ([MS-OXPROPS] section 2.672) (3003001F [Unicode])
BA 00 00 00	length 186 (0xBA)
2F 00 4F 00-3D 00 46 00 49 00 52 00-53 00 54 00	varSizeValue /.O.=.F. I.R.S.T.

Bytes on the wire	Value/description
20 00 4F 00-52 00 47 00 41 00 4E 00-49 00 5A 00 41 00 54 00-49 00 4F 00	.O.R.G. A.N.I.Z. A.T.I.O.
... value truncated...	
1F 00 01 30	propDef **PidTagDisplayName** property ([MS-OXCFOLD] section 2.2.2.2.2.5) (3001001F [Unicode])
06 00 00 00	length 6 (0x6)
74 00 31 00-00 00	varSizeValue t.1...
02 01 F6 0F	propDef **PidTagInstanceKey** property ([MS-OXPROPS] section 2.734) (0FF60102 [Binary])
04 00 00 00	length 4 (0x4)
00 00 00 00	varSizeValue ....
03 00 15 0C	propDef **PidTagRecipientType** property ([MS-OXOMSG] section 2.2.3.1) (0C150003 [Int32])
01 00 00 00	fixedSizeValue [Int32] 1
02 01 FF 0F	propDef **PidTagEntryId** property ([MS-OXPROPS] section 2.674) (0FFF0102 [Binary])
79 00 00 00	length 121 (0x79)
00 00 00 00-DC A7 40 C8 C0 42 10 1A-B4 B9 08 00 2B 2F E1 82-01 00 00 00 00 00 00 00-2F 4F 3D 46 49 52 53 54-20 4F 52 47	varSizeValue .....@. .B..... +/..... ..../O=F IRST ORG
... value truncated...	
02 01 0B 30	propDef **PidTagSearchKey** property (300B0102 [Binary])
60 00 00 00	length

Bytes on the wire	Value/description
	96 (0x60)
45 58 3A 2F-4F 3D 46 49 52 53 54 20-4F 52 47 41 4E 49 5A 41-54 49 4F 4E 2F 4F 55 3D-45 58 43 48	varSizeValue EX:/O=FI RST ORGA NIZATION /OU=EXCH
... value truncated...	
1F 00 20 3A	propDef **PidTagTransmittableDisplayName** property ([MS-OXOABK] section 2.2.3.8) (3A20001F [Unicode])
06 00 00 00	length 6 (0x6)
74 00 31 00-00 00	varSizeValue t.1...
0B 00 0F 0E	propDef **PidTagResponsibility** property ([MS-OXPROPS] section 2.922) (0E0F000B [Bool])
01 00	fixedSizeValue [Bool] **TRUE**
0B 00 40 3A	propDef **PidTagSendRichInfo** property (3A40000B [Bool])
01 00	fixedSizeValue [Bool] **TRUE**
03 00 FD 5F	propDef **PidTagRecipientFlags** property ([MS-OXOCAL] section 2.2.4.10.1) (5FFD0003 [Int32])
01 00 00 00	fixedSizeValue [Int32] 1
02 01 F7 5F	propDef **PidTagRecipientEntryId** property ([MS-OXPROPS] section 2.891) (5FF70102 [Binary])
79 00 00 00	length 121 (0x79)
00 00 00 00-DC A7 40 C8 C0 42 10 1A-B4 B9 08 00 2B 2F E1 82-01 00 00 00 00 00 00 00-2F 6F 3D 46 69 72 73 74-	varSizeValue .....@. .B..... +/..... ..../o=F irst Org

Bytes on the wire	Value/description
20 4F 72 67	
... value truncated...	
1F 00 FE 39	propDef **PidTagSmtpAddress** property ([MS-OXPROPS] section 2.1010) (39FE001F [Unicode])
46 00 00 00	length 70 (0x46)
74 00 31 00-40 00 65 00 75 00 6D 00-61 00 72 00 75 00 2D 00-64 00 6F 00 6D 00 2E 00-65 00 78 00 74 00 65 00-73 00 74 00	varSizeValue t.1.@.e. u.m.a.r. u.-.d.o. m...e.x. t.e.s.t.
... value truncated...	
03 00 05 39	propDef **PidTagDisplayTypeEx** property ([MS-OXOABK] section 2.2.3.12) (39050003 [Int32])
00 00 00 40	fixedSizeValue [Int32] 1073741824
03 00 00 39	propDef **PidTagDisplayType** property ([MS-OXOABK] section 2.2.3.11) (39000003 [Int32])
00 00 00 00	fixedSizeValue [Int32] 0
03 00 FE 0F	propDef **PidTagObjectType** property ([MS-OXOABK] section 2.2.3.10) (0FFE0003 [Int32])
06 00 00 00	fixedSizeValue [Int32] 6
1F 00 FF 39	propDef **PidTagAddressBookDisplayNamePrintable** property ([MS-OXOABK] section 2.2.3.7) (39FF001F [Unicode])
06 00 00 00	length 6 (0x6)
74 00 31 00-00 00	varSizeValue t.1...
1F 00 00 3A	propDef **PidTagAccount** property ([MS-OXOABK] section 2.2.3.20) (3A00001F [Unicode])
06 00 00 00	length 6 (0x6)

Bytes on the wire	Value/description
74 00 31 00-00 00	varSizeValue t.1...
03 00 FF 5F	propDef **PidTagRecipientTrackStatus** property ([MS-OXOCAL] section 2.2.4.10.2) (5FFF0003 [Int32])
00 00 00 00	fixedSizeValue [Int32] 0
03 00 DE 5F	propDef Unspecified property (5FDE0003 [Int32])
00 00 00 00	fixedSizeValue [Int32] 0
1F 00 F6 5F	propDef **PidTagRecipientDisplayName** property ([MS-OXPROPS] section 2.890) (5FF6001F [Unicode])
06 00 00 00	length 6 (0x6)
74 00 31 00-00 00	varSizeValue t.1...
03 00 DF 5F	propDef **PidTagRecipientOrder** property ([MS-OXPROPS] section 2.893) (5FDF0003 [Int32])
00 00 00 00	fixedSizeValue [Int32] 0
03 00 04 40	**marker** **EndToRecip marker (section 2.2.4.1.4) (40040003 [Int32])**
03 00 16 40	propDef **MetaTagFXDelProp** property (40160003 [Int32])
0D 00 13 0E	FixedSizeValue **PidTagMessageAttachments** property ([MS-OXPROPS] section 2.776) (0E13000D [Object])
03 00 00 40	**marker** **NewAttach marker (section 2.2.4.1.4) (40000003 [Int32])**
03 00 21 0E	propDef **PidTagAttachNumber** property ([MS-OXCMSG] section 2.2.2.6) (0E210003 [Int32])
00 00 00 00	marker [Int32] 0
02 01 02 37	propDef **PidTagAttachEncoding** property ([MS-OXCMSG] section 2.2.2.20) (37020102 [Binary])
00 00 00 00	length 0 (0x0)
03 00 0B 37	propDef **PidTagRenderingPosition** property ([MS-OXCMSG] section 2.2.2.16) (370B0003 [Int32])

Bytes on the wire	Value/description
FF FF FF FF	fixedSizeValue [Int32] -1
03 00 20 0E	propDef **PidTagAttachSize** property ([MS-OXCMSG] section 2.2.2.5) (0E200003 [Int32])
E7 15 00 00	fixedSizeValue [Int32] 5607
03 00 F7 0F	propDef **PidTagAccessLevel** property (0FF70003 [Int32])
00 00 00 00	fixedSizeValue [Int32] 0
40 00 07 30	propDef **PidTagCreationTime** property (30070040 [SysTime])
E2 EA E3 B1-BC 84 C8 01	fixedSizeValue [SysTime] 2008-03-13T03:45:35.3281250
40 00 08 30	propDef **PidTagLastModificationTime** property (30080040 [SysTime])
E2 EA E3 B1-BC 84 C8 01	fixedSizeValue [SysTime] 2008-03-13T03:45:35.3281250
03 00 05 37	propDef **PidTagAttachMethod** property ([MS-OXCMSG] section 2.2.2.9) (37050003 [Int32])
05 00 00 00	fixedSizeValue [Int32] 5
02 01 09 37	propDef **PidTagAttachRendering** property ([MS-OXCMSG] section 2.2.2.17) (37090102 [Binary])
B8 0D 00 00	length 3512 (0xDB8)
01 00 09 00-00 03 DC 06 00 00 00 00-21 06 00 00 00 00 05 00-00 00 09 02 00 00 00 00-05 00 00 00 01 02 FF FF-FF 00 A5 00	varSizeValue ....!... ....
... value truncated...	
03 00 14 37	propDef **PidTagAttachFlags** property ([MS-OXCMSG] section 2.2.2.18) (37140003 [Int32])
00 00 00 00	fixedSizeValue

Bytes on the wire	Value/description
	[Int32] 0
0B 00 FE 7F	propDef **PidTagAttachmentHidden** property ([MS-OXCMSG] section 2.2.2.24) (7FFE000B [Bool])
00 00	fixedSizeValue [Bool] **FALSE**
1F 00 04 37	propDef **PidTagAttachFilename** property ([MS-OXCMSG] section 2.2.2.11) (3704001F [Unicode])
0E 00 00 00	length 14 (0xE)
54 00 65 00-73 00 74 00 20 00 31 00-00 00	varSizeValue T.e.s.t. .1...
0B 00 FF 7F	propDef **PidTagAttachmentContactPhoto** property ([MS-OXPROPS] section 2.588) (7FFF000B [Bool])
00 00	fixedSizeValue [Bool] **FALSE**
1F 00 01 30	propDef **PidTagDisplayName** property (3001001F [Unicode])
0E 00 00 00	length 14 (0xE)
54 00 65 00-73 00 74 00 20 00 31 00-00 00	varSizeValue T.e.s.t. .1...
02 01 F9 0F	propDef **PidTagRecordKey** property ([MS-OXPROPS] section 2.901) (0FF90102 [Binary])
04 00 00 00	length 4 (0x4)
00 00 00 00	varSizeValue ....
03 00 01 40	**marker** **StartEmbed** marker (section 2.2.4.1.4) (40010003 [Int32])
14 00 4A 67	propDef **PidTagMid** property (section 2.2.1.2.1) (674A0014 [Int64])
01 00 00 00-00 78 48 C1	fixedSizeValue [Int64] -4519230284670959615
0B 00 02 00	propDef **PidTagAlternateRecipientAllowed** property (0002000B [Bool])

Bytes on the wire	Value/description
01 00	fixedSizeValue [Bool] **TRUE**
03 00 17 00	propDef **PidTagImportance** property (00170003 [Int32])
01 00 00 00	fixedSizeValue [Int32] 1
1F 00 1A 00	propDef **PidTagMessageClass** property (001A001F [Unicode])
12 00 00 00	length 18 (0x12)
49 00 50 00-4D 00 2E 00 4E 00 6F 00-74 00 65 00 00 00	varSizeValue I.P.M... N.o.t.e. ..
0B 00 23 00	propDef **PidTagOriginatorDeliveryReportRequested** property (0023000B [Bool])
00 00	fixedSizeValue [Bool] **FALSE**
03 00 26 00	propDef **PidTagPriority** property (00260003 [Int32])
00 00 00 00	fixedSizeValue [Int32] 0
0B 00 29 00	propDef **PidTagReadReceiptRequested** property (0029000B [Bool])
00 00	fixedSizeValue [Bool] **FALSE**
03 00 36 00	propDef **PidTagSensitivity** property (00360003 [Int32])
00 00 00 00	fixedSizeValue [Int32] 0
1F 00 37 00	propDef **PidTagSubject** property (0037001F [Unicode])
0E 00 00 00	length 14 (0xE)
54 00 65 00-73 00 74 00 20 00 31 00-00 00	varSizeValue T.e.s.t. .1...

Bytes on the wire	Value/description
40 00 39 00	propDef **PidTagClientSubmitTime** property (00390040 [SysTime])
00 B4 A1 9D-8B 84 C8 01	fixedSizeValue [SysTime] 2008-03-12T21:54:16.0000000
02 01 3B 00	propDef **PidTagSentRepresentingSearchKey** property (003B0102 [Binary])
60 00 00 00	length 96 (0x60)
45 58 3A 2F-4F 3D 46 49 52 53 54 20-4F 52 47 41 4E 49 5A 41-54 49 4F 4E 2F 4F 55 3D-45 58 43 48 41 4E 47 45-20 41 44 4D	varSizeValue EX:/O=FI RST ORGA NIZATION /OU=EXCH ANGE ADM
... value truncated...	
1F 00 3D 00	propDef **PidTagSubjectPrefix** property (003D001F [Unicode])
02 00 00 00	length 2 (0x2)
00 00	varSizeValue ..
02 01 3F 00	propDef **PidTagReceivedByEntryId** property (003F0102 [Binary])
79 00 00 00	length 121 (0x79)
00 00 00 00-DC A7 40 C8 C0 42 10 1A-B4 B9 08 00 2B 2F E1 82-01 00 00 00 00 00 00 00-2F 4F 3D 46 49 52 53 54-20 4F 52 47	varSizeValue .....@. .B..... +/..... ..../O=F IRST ORG
... value truncated...	
1F 00 40 00	propDef **PidTagReceivedByName** property (0040001F [Unicode])

Bytes on the wire	Value/description
06 00 00 00	length 6 (0x6)
74 00 31 00-00 00	varSizeValue t.1...
02 01 41 00	propDef **PidTagSentRepresentingEntryId** property (00410102 [Binary])
79 00 00 00	length 121 (0x79)
00 00 00 00-DC A7 40 C8 C0 42 10 1A-B4 B9 08 00 2B 2F E1 82-01 00 00 00 00 00 00 00-2F 4F 3D 46 49 52 53 54-20 4F 52 47	varSizeValue .....@. .B..... +/..... ..../O=F IRST ORG
... value truncated...	
1F 00 42 00	propDef **PidTagSentRepresentingName** property (0042001F [Unicode])
06 00 00 00	length 6 (0x6)
74 00 31 00-00 00	varSizeValue t.1...
02 01 43 00	propDef **PidTagReceivedRepresentingEntryId** property (00430102 [Binary])
79 00 00 00	length 121 (0x79)
00 00 00 00-DC A7 40 C8 C0 42 10 1A-B4 B9 08 00 2B 2F E1 82-01 00 00 00 00 00 00 00-2F 4F 3D 46 49 52 53 54-20 4F 52 47	varSizeValue .....@. .B..... +/..... ..../O=F IRST ORG
... value truncated...	
1F 00 44 00	propDef **PidTagReceivedRepresentingName** property (0044001F [Unicode])

Bytes on the wire	Value/description
06 00 00 00	length 6 (0x6)
74 00 31 00- 00 00	varSizeValue t.1...
02 01 51 00	propDef **PidTagReceivedBySearchKey** property (00510102 [Binary])
60 00 00 00	length 96 (0x60)
45 58 3A 2F- 4F 3D 46 49 52 53 54 20- 4F 52 47 41 4E 49 5A 41- 54 49 4F 4E 2F 4F 55 3D- 45 58 43 48 41 4E 47 45- 20 41 44 4D	varSizeValue EX:/O=FI RST ORGA NIZATION /OU=EXCH ANGE ADM
... value truncated...	
02 01 52 00	propDef **PidTagReceivedRepresentingSearchKey** property (00520102 [Binary])
60 00 00 00	length 96 (0x60)
45 58 3A 2F- 4F 3D 46 49 52 53 54 20- 4F 52 47 41 4E 49 5A 41- 54 49 4F 4E 2F 4F 55 3D- 45 58 43 48 41 4E 47 45- 20 41 44 4D	varSizeValue EX:/O=FI RST ORGA NIZATION /OU=EXCH ANGE ADM
... value truncated...	
0B 00 63 00	propDef **PidTagResponseRequested** property ([MS-OXOMSG] section 2.2.1.46) (0063000B [Bool])
01 00	fixedSizeValue [Bool] **TRUE**
1F 00 64 00	propDef **PidTagSentRepresentingAddressType** property (0064001F [Unicode])
06 00 00 00	length 6 (0x6)

Bytes on the wire	Value/description
45 00 58 00-00 00	varSizeValue E.X...
1F 00 65 00	propDef **PidTagSentRepresentingEmailAddress** property (0065001F [Unicode])
BA 00 00 00	length 186 (0xBA)
2F 00 4F 00-3D 00 46 00 49 00 52 00-53 00 54 00 20 00 4F 00-52 00 47 00 41 00 4E 00-49 00 5A 00 41 00 54 00-49 00 4F 00	varSizeValue /.O.=.F. I.R.S.T. .O.R.G. A.N.I.Z. A.T.I.O.
... value truncated...	
1F 00 70 00	propDef **PidTagConversationTopic** property (0070001F [Unicode])
0E 00 00 00	length 14 (0xE)
54 00 65 00-73 00 74 00 20 00 31 00-00 00	varSizeValue T.e.s.t. .1...
02 01 71 00	propDef **PidTagConversationIndex** property (00710102 [Binary])
16 00 00 00	length 22 (0x16)
01 C8 84 8B-9D B1 08 58 53 52 00 5B-4A D4 96 BA 3C 88 9D B4-16 AE	varSizeValue ....X SR.[J... <.....
1F 00 75 00	propDef **PidTagReceivedByAddressType** property (0075001F [Unicode])
06 00 00 00	length 6 (0x6)
45 00 58 00-00 00	varSizeValue E.X...
1F 00 76 00	propDef

Bytes on the wire	Value/description
	PidTagReceivedByEmailAddress property (0076001F [Unicode])
BA 00 00 00	length 186 (0xBA)
2F 00 4F 00- 3D 00 46 00 49 00 52 00- 53 00 54 00 20 00 4F 00- 52 00 47 00 41 00 4E 00- 49 00 5A 00 41 00 54 00- 49 00 4F 00	varSizeValue /.O.=.F. I.R.S.T. .O.R.G. A.N.I.Z. A.T.I.O.
... value truncated...	
1F 00 77 00	propDef **PidTagReceivedRepresentingAddressType** property (0077001F [Unicode])
06 00 00 00	length 6 (0x6)
45 00 58 00- 00 00	varSizeValue E.X...
1F 00 78 00	propDef **PidTagReceivedRepresentingEmailAddress** property (0078001F [Unicode])
BA 00 00 00	length 186 (0xBA)
2F 00 4F 00- 3D 00 46 00 49 00 52 00- 53 00 54 00 20 00 4F 00- 52 00 47 00 41 00 4E 00- 49 00 5A 00 41 00 54 00- 49 00 4F 00	varSizeValue /.O.=.F. I.R.S.T. .O.R.G. A.N.I.Z. A.T.I.O.
... value truncated...	
1F 00 7D 00	propDef **PidTagTransportMessageHeaders** property (007D001F [Unicode])
B0 06 00 00	length 1712 (0x6B0)
52 00 65 00- 63 00 65 00 69 00 76 00- 65 00 64 00 3A 00 20 00-	varSizeValue R.e.c.e. i.v.e.d. :..f.r.

Bytes on the wire	Value/description
66 00 72 00 6F 00 6D 00- 20 00 45 00 58 00 43 00- 48 00 2D 00	o.m..E. X.C.H.-.
... value truncated...	
0B 00 17 0C	propDef **PidTagReplyRequested** property ([MS-OXOMSG] section 2.2.1.45) (0C17000B [Bool])
01 00	fixedSizeValue [Bool] **TRUE**
02 01 19 0C	propDef **PidTagSenderEntryId** property (0C190102 [Binary])
79 00 00 00	length 121 (0x79)
00 00 00 00- DC A7 40 C8 C0 42 10 1A- B4 B9 08 00 2B 2F E1 82- 01 00 00 00 00 00 00 00- 2F 4F 3D 46 49 52 53 54- 20 4F 52 47	varSizeValue .....@. .B..... +/..... ..../O=F IRST ORG
... value truncated...	
1F 00 1A 0C	propDef **PidTagSenderName** property (0C1A001F [Unicode])
06 00 00 00	length 6 (0x6)
74 00 31 00- 00 00	varSizeValue t.1...
02 01 1D 0C	propDef **PidTagSenderSearchKey** property (0C1D0102 [Binary])
60 00 00 00	length 96 (0x60)
45 58 3A 2F- 4F 3D 46 49 52 53 54 20- 4F 52 47 41 4E 49 5A 41- 54 49 4F 4E 2F 4F 55 3D- 45 58 43 48	varSizeValue EX:/O=FI RST ORGA NIZATION /OU=EXCH ANGE ADM

Bytes on the wire	Value/description
41 4E 47 45- 20 41 44 4D	
... value truncated...	
1F 00 1E 0C	propDef **PidTagSenderAddressType** property (0C1E001F [Unicode])
06 00 00 00	length 6 (0x6)
45 00 58 00- 00 00	varSizeValue E.X...
1F 00 1F 0C	propDef **PidTagSenderEmailAddress** property (0C1F001F [Unicode])
BA 00 00 00	length 186 (0xBA)
2F 00 4F 00- 3D 00 46 00 49 00 52 00- 53 00 54 00 20 00 4F 00- 52 00 47 00 41 00 4E 00- 49 00 5A 00 41 00 54 00- 49 00 4F 00	varSizeValue /.O.=.F. I.R.S.T. .O.R.G. A.N.I.Z. A.T.I.O.
... value truncated...	
1F 00 D4 83- 03 20 06 00 00 00 00 00- C0 00 00 00 00 00 00 46- 00 27 81 00 00	propDef **PidLidTaskRole** property (0x8127 [PSETID_Task]) [Unicode]
02 00 00 00	length 2 (0x2)
00 00	varSizeValue ..
03 00 D3 83- 03 20 06 00 00 00 00 00- C0 00 00 00 00 00 00 46- 00 2A 81 00 00	propDef **PidLidTaskAcceptanceState** property (0x812A [PSETID_Task]) [Int32]
00 00 00 00	fixedSizeValue

Bytes on the wire	Value/description
	[Int32] 0
0B 00 D2 83-03 20 06 00 00 00 00 00-C0 00 00 00 00 00 00 46-00 2C 81 00 00	propDef **PidLidTaskFFixOffline** property (0x812C [PSETID_Task]) [Bool]
00 00	fixedSizeValue [Bool] **FALSE**
40 00 06 0E	propDef **PidTagMessageDeliveryTime** property (0E060040 [SysTime])
00 0E 04 A0-8B 84 C8 01	fixedSizeValue [SysTime] 2008-03-12T21:54:20.0000000
03 00 07 0E	propDef **PidTagMessageFlags** property (0E070003 [Int32])
01 00 00 00	fixedSizeValue [Int32] 1
03 00 CF 83-03 20 06 00 00 00 00 00-C0 00 00 00 00 00 00 46-00 10 81 00 00	propDef **PidLidTaskActualEffort** property (0x8110 [PSETID_Task]) [Int32]
00 00 00 00	fixedSizeValue [Int32] 0
03 00 17 0E	propDef **PidTagMessageStatus** property (0E170003 [Int32])
00 00 00 00	fixedSizeValue [Int32] 0
0B 00 D1 83-03 20 06 00 00 00 00 00-C0 00 00 00 00 00 00 46-00 24 81 00 00	propDef **PidLidTaskNoCompute** property (0x8124 [PSETID_Task]) [Bool]
00 00	fixedSizeValue [Bool] **FALSE**
1F 00 1D 0E	propDef **PidTagNormalizedSubject** property (0E1D001F [Unicode])

Bytes on the wire	Value/description
0E 00 00 00	length 14 (0xE)
54 00 65 00-73 00 74 00 20 00 31 00-00 00	varSizeValue T.e.s.t. .1...
0B 00 1F 0E	propDef **PidTagRtfInSync** property (0E1F000B [Bool])
01 00	fixedSizeValue [Bool] **TRUE**
03 00 23 0E	propDef Unspecified property (0E230003 [Int32])
1B 00 00 00	fixedSizeValue [Int32] 27
03 00 2B 0E	propDef **PidTagToDoItemFlags** property ([MS-OXOFLAG] section 2.2.1.6) (0E2B0003 [Int32])
01 00 00 00	fixedSizeValue [Int32] 1
03 00 79 0E	propDef **PidTagTrustSender** property (0E790003 [Int32])
01 00 00 00	fixedSizeValue [Int32] 1
03 00 D0 83-03 20 06 00 00 00 00 00-C0 00 00 00 00 00 00 46-00 11 81 00 00	propDef **PidLidTaskEstimatedEffort** property (0x8111 [PSETID_Task]) [Int32]
00 00 00 00	fixedSizeValue [Int32] 0
03 00 F7 0F	propDef **PidTagAccessLevel** property (0FF70003 [Int32])
00 00 00 00	fixedSizeValue [Int32] 0
0B 00 D6 83-03 20 06 00 00 00 00 00-C0 00 00 00 00 00 00 46-00 26 81 00 00	propDef **PidLidTaskFRecurring** property (0x8126 [PSETID_Task]) [Bool]

Bytes on the wire	Value/description
00 00	fixedSizeValue [Bool] **FALSE**
02 01 09 10	propDef **PidTagRtfCompressed** property ([MS-OXCMSG] section 2.2.1.56.4) (10090102 [Binary])
22 05 00 00	length 1314 (0x522)
1E 05 00 00-85 0B 00 00 4C 5A 46 75-31 AE 9B E3 03 00 0A 00-72 63 70 67 31 32 35 83-00 50 03 52 68 74 6D 6C-31 03 31 F8	varSizeValue ..... LZFu1... ....rcpg 125..P.R html1.1.
... value truncated...	
0B 00 D5 83-03 20 06 00 00 00 00 00-C0 00 00 00 00 00 00 46-00 03 81 00 00	propDef **PidLidTeamTask** property (0x8103 [PSETID_Task]) [Bool]
00 00	fixedSizeValue [Bool] **FALSE**
1F 00 35 10	propDef **PidTagInternetMessageId** property (1035001F [Unicode])
AC 00 00 00	length 172 (0xAC)
3C 00 31 00-39 00 44 00 37 00 46 00-42 00 30 00 46 00 30 00-36 00 31 00 36 00 41 00-31 00 34 00 31 00 42 00-46 00 46 00	varSizeValue <.1.9.D. 7.F.B.0. F.0.6.1. 6.A.1.4. 1.B.F.F.
... value truncated...	
03 00 80 10	propDef **PidTagIconIndex** property (10800003 [Int32])
FF FF FF FF	fixedSizeValue

Bytes on the wire	Value/description
	[Int32] -1
03 00 90 10	propDef **PidTagFlagStatus** property ([MS-OXOFLAG] section 2.2.1.1) (10900003 [Int32])
02 00 00 00	fixedSizeValue [Int32] 2
03 00 95 10	propDef **PidTagFollowupIcon** property ([MS-OXOFLAG] section 2.2.1.2) (10950003 [Int32])
06 00 00 00	fixedSizeValue [Int32] 6
40 00 07 30	propDef **PidTagCreationTime** property (30070040 [SysTime])
90 F8 65 B0-BC 84 C8 01	fixedSizeValue [SysTime] 2008-03-13T03:45:32.8250000
40 00 08 30	propDef **PidTagLastModificationTime** property (30080040 [SysTime])
90 F8 65 B0-BC 84 C8 01	fixedSizeValue [SysTime] 2008-03-13T03:45:32.8250000
02 01 0B 30	propDef **PidTagSearchKey** property (300B0102 [Binary])
10 00 00 00	length 16 (0x10)
87 56 4A B2-FC C2 77 46 A4 81 15 08-9D 47 46 8C	varSizeValue .VJ...wF .....GF.
02 01 10 30	propDef **PidTagTargetEntryId** property ([MS-OXOMSG] section 2.2.1.76) (30100102 [Binary])
46 00 00 00	length 70 (0x46)
00 00 00 00-FE C7 EE E9 76 05 2D 4F-80 00 61 68 94 97 4B 0A-07 00 19 D7 FB 0F 06 16-A1 41 BF F6 91 C7 63 DA-A8 66 00 00	varSizeValue ..... v.-O..ah ..K..... .....A.. ..c..f..
... value truncated...	
0B 00 40 3A	propDef

Bytes on the wire	Value/description
	PidTagSendRichInfo property (3A40000B [Bool])
01 00	fixedSizeValue [Bool] **TRUE**
03 00 DE 3F	propDef **PidTagInternetCodepage** property (3FDE0003 [Int32])
9F 4E 00 00	fixedSizeValue [Int32] 20127
03 00 F1 3F	propDef **PidTagMessageLocaleId** property (3FF10003 [Int32])
09 04 00 00	fixedSizeValue [Int32] 1033
1F 00 F8 3F	propDef **PidTagCreatorName** property ([MS-OXPROPS] section 2.647) (3FF8001F [Unicode])
06 00 00 00	length 6 (0x6)
74 00 31 00-00 00	varSizeValue t.1...
1F 00 FA 3F	propDef **PidTagLastModifierName** property ([MS-OXCPRPT] section 2.2.1.5) (3FFA001F [Unicode])
06 00 00 00	length 6 (0x6)
74 00 31 00-00 00	varSizeValue t.1...
03 00 FD 3F	propDef **PidTagMessageCodepage** property (3FFD0003 [Int32])
E3 04 00 00	fixedSizeValue [Int32] 1251
03 00 19 40	propDef Unspecified property (40190003 [Int32])
00 00 00 00	fixedSizeValue [Int32] 0
03 00 1A 40	propDef **PidTagSentRepresentingFlags** property (401A0003 [Int32])
00 00 00 00	fixedSizeValue [Int32] 0
03 00 1B 40	propDef Unspecified property (401B0003 [Int32])

Bytes on the wire	Value/description
00 00 00 00	fixedSizeValue [Int32] 0
03 00 1C 40	propDef Unspecified property (401C0003 [Int32])
00 00 00 00	fixedSizeValue [Int32] 0
03 00 76 40	propDef **PidTagContentFilterSpamConfidenceLevel** property (40760003 [Int32])
FF FF FF FF	fixedSizeValue [Int32] -1
03 00 02 59	propDef **PidTagInternetMailOverrideFormat** property (59020003 [Int32])
00 00 16 00	fixedSizeValue [Int32] 1441792
03 00 09 59	propDef **PidTagMessageEditorFormat** property (59090003 [Int32])
02 00 00 00	fixedSizeValue [Int32] 2
0B 00 4A 66	propDef **PidTagHasNamedProperties** property ([MS-OXPROPS] section 2.709) (664A000B [Bool])
01 00	fixedSizeValue [Bool] **TRUE**
03 00 02 80- 08 20 06 00 00 00 00 00- C0 00 00 00 00 00 00 46- 00 10 85 00 00	propDef **PidLidSideEffects** property (0x8510 [PSETID_Common]) [Int32]
00 00 00 00	fixedSizeValue [Int32] 0
0B 00 08 80- 08 20 06 00 00 00 00 00- C0 00 00 00 00 00 00 46- 00 03 85 00 00	propDef **PidLidReminderSet** property (0x8503 [PSETID_Common]) [Bool]
00 00	fixedSizeValue [Bool] **FALSE**
1F 00 1A 80-	propDef

Bytes on the wire	Value/description
08 20 06 00 00 00 00 00-C0 00 00 00 00 00 00 46-00 A4 85 00 00	**PidLidToDoTitle** property ([MS-OXOFLAG] section 2.2.1.12) (0x85A4 [PSETID_Common]) [Unicode]
0E 00 00 00	length 14 (0xE)
54 00 65 00-73 00 74 00 20 00 31 00-00 00	varSizeValue T.e.s.t. .1...
1F 00 2C 80-08 20 06 00 00 00 00 00-C0 00 00 00 00 00 00 46-00 30 85 00 00	propDef **PidLidFlagRequest** property ([MS-OXOFLAG] section 2.2.1.9) (0x8530 [PSETID_Common]) [Unicode]
14 00 00 00	length 20 (0x14)
46 00 6F 00-6C 00 6C 00 6F 00 77 00-20 00 75 00 70 00 00 00	varSizeValue F.o.l.l. o.w..u. p...
0B 00 4D 81-08 20 06 00 00 00 00 00-C0 00 00 00 00 00 00 46-00 0E 85 00 00	propDef **PidLidAgingDontAgeMe** property (0x850E [PSETID_Common]) [Bool]
00 00	fixedSizeValue [Bool] **FALSE**
03 00 84 81-08 20 06 00 00 00 00 00-C0 00 00 00 00 00 00 46-00 18 85 00 00	propDef **PidLidTaskMode** property (0x8518 [PSETID_Common]) [Int32]
00 00 00 00	fixedSizeValue [Int32] 0
0B 00 4B 82-08 20 06 00	propDef **PidLidPrivate** property (0x8506 [PSETID_Common]) [Bool]

Bytes on the wire	Value/description
00 00 00 00- C0 00 00 00 00 00 00 46- 00 06 85 00 00	
00 00	fixedSizeValue [Bool] **FALSE**
0B 00 4F 82- 08 20 06 00 00 00 00 00- C0 00 00 00 00 00 00 46- 00 82 85 00 00	propDef **PidLidUseTnef** property (0x8582 [PSETID_Common]) [Bool]
00 00	fixedSizeValue [Bool] **FALSE**
40 00 68 82- 08 20 06 00 00 00 00 00- C0 00 00 00 00 00 00 46- 00 A0 85 00 00	propDef **PidLidToDoOrdinalDate** property ([MS-OXOFLAG] section 2.2.1.13) (0x85A0 [PSETID_Common]) [SysTime]
F0 55 C3 C6- 8B 84 C8 01	fixedSizeValue [SysTime] 2008-03-12T21:55:25.0070000
1F 00 69 82- 08 20 06 00 00 00 00 00- C0 00 00 00 00 00 00 46- 00 A1 85 00 00	propDef **PidLidToDoSubOrdinal** property ([MS-OXOFLAG] section 2.2.1.14) (0x85A1 [PSETID_Common]) [Unicode]
10 00 00 00	length 16 (0x10)
35 00 35 00- 35 00 35 00 35 00 35 00- 35 00 00 00	varSizeValue 5.5.5.5. 5.5.5...
03 00 A8 83- 08 20 06 00 00 00 00 00- C0 00 00 00 00 00 00 46- 00 01 85 00 00	propDef **PidLidReminderDelta** property (0x8501 [PSETID_Common]) [Int32]
00 00 00 00	fixedSizeValue

Bytes on the wire	Value/description
	[Int32] 0
40 00 A9 83-03 20 06 00 00 00 00 00-C0 00 00 00 00 00 00 46-00 05 81 00 00	propDef **PidLidTaskDueDate** property ([MS-OXOTASK] section 2.2.2.2.5) (0x8105 [PSETID_Task]) [SysTime]
00 00 CB 03-D4 83 C8 01	fixedSizeValue [SysTime] 2008-03-12T00:00:00.0000000
40 00 AA 83-03 20 06 00 00 00 00 00-C0 00 00 00 00 00 00 46-00 04 81 00 00	propDef **PidLidTaskStartDate** property ([MS-OXOTASK] section 2.2.2.2.4) (0x8104 [PSETID_Task]) [SysTime]
00 00 CB 03-D4 83 C8 01	fixedSizeValue [SysTime] 2008-03-12T00:00:00.0000000
40 00 AB 83-08 20 06 00 00 00 00 00-C0 00 00 00 00 00 00 46-00 16 85 00 00	propDef **PidLidCommonStart** property ([MS-OXOTASK] section 2.2.2.1.3) (0x8516 [PSETID_Common]) [SysTime]
00 D8 29 B0-0E 84 C8 01	fixedSizeValue [SysTime] 2008-03-12T07:00:00.0000000
40 00 AC 83-08 20 06 00 00 00 00 00-C0 00 00 00 00 00 00 46-00 17 85 00 00	propDef **PidLidCommonEnd** property ([MS-OXOTASK] section 2.2.2.1.4) (0x8517 [PSETID_Common]) [SysTime]
00 D8 29 B0-0E 84 C8 01	fixedSizeValue [SysTime] 2008-03-12T07:00:00.0000000
03 00 AD 83-03 20 06 00 00 00 00 00-C0 00 00 00 00 00 00 46-00 01 81 00 00	propDef **PidLidTaskStatus** property ([MS-OXOTASK] section 2.2.2.2.2) (0x8101 [PSETID_Task]) [Int32]
00 00 00 00	fixedSizeValue [Int32] 0

Bytes on the wire	Value/description
05 00 AE 83-03 20 06 00 00 00 00 00-C0 00 00 00 00 00 00 46-00 02 81 00 00	propDef **PidLidPercentComplete** property (0x8102 [PSETID_Task]) [Double]
00 00 00 00-00 00 00 00	fixedSizeValue [Double] 0
0B 00 B0 83-03 20 06 00 00 00 00 00-C0 00 00 00 00 00 00 46-00 1C 81 00 00	propDef **PidLidTaskComplete** property (0x811C [PSETID_Task]) [Bool]
00 00	fixedSizeValue [Bool] **FALSE**
03 00 CA 83-03 20 06 00 00 00 00 00-C0 00 00 00 00 00 00 46-00 13 81 00 00	propDef **PidLidTaskState** property (0x8113 [PSETID_Task]) [Int32]
01 00 00 00	fixedSizeValue [Int32] 1
03 00 CB 83-03 20 06 00 00 00 00 00-C0 00 00 00 00 00 00 46-00 12 81 00 00	propDef **PidLidTaskVersion** property (0x8112 [PSETID_Task]) [Int32]
01 00 00 00	fixedSizeValue [Int32] 1
03 00 CC 83-03 20 06 00 00 00 00 00-C0 00 00 00 00 00 00 46-00 23 81 00 00	propDef **PidLidTaskOrdinal** property (0x8123 [PSETID_Task]) [Int32]
FF FF FF 7F	fixedSizeValue [Int32] 2147483647

Bytes on the wire	Value/description
1F 00 CD 83- 03 20 06 00 00 00 00 00- C0 00 00 00 00 00 00 46- 00 21 81 00 00	propDef **PidLidTaskAssigner** property (0x8121 [PSETID_Task]) [Unicode]
02 00 00 00	length 2 (0x2)
00 00	varSizeValue ..
03 00 CE 83- 03 20 06 00 00 00 00 00- C0 00 00 00 00 00 00 46- 00 29 81 00 00	propDef **PidLidTaskOwnership** property (0x8129 [PSETID_Task]) [Int32]
00 00 00 00	fixedSizeValue [Int32] 0
03 00 03 40	**marker** **StartRecip** marker (section 2.2.4.1.4) (40030003 [Int32])
03 00 00 30	propDef **PidTagRowid** property (30000003 [Int32])
00 00 00 00	fixedSizeValue [Int32] 0
1F 00 02 30	propDef **PidTagAddressType** property (3002001F [Unicode])
06 00 00 00	length 6 (0x6)
45 00 58 00- 00 00	varSizeValue E.X...
1F 00 03 30	propDef **PidTagEmailAddress** property (3003001F [Unicode])
BA 00 00 00	length 186 (0xBA)
2F 00 4F 00- 3D 00 46 00 49 00 52 00- 53 00 54 00 20 00 4F 00- 52 00 47 00 41 00 4E 00-	varSizeValue /.O.=.F. I.R.S.T. .O.R.G. A.N.I.Z. A.T.I.O.

Bytes on the wire	Value/description
49 00 5A 00 41 00 54 00- 49 00 4F 00	
... value truncated...	
1F 00 01 30	propDef **PidTagDisplayName** property (3001001F [Unicode])
06 00 00 00	length 6 (0x6)
74 00 31 00- 00 00	varSizeValue t.1...
02 01 F6 0F	propDef **PidTagInstanceKey** property (0FF60102 [Binary])
04 00 00 00	length 4 (0x4)
00 00 00 00	varSizeValue ....
03 00 15 0C	propDef **PidTagRecipientType** property (0C150003 [Int32])
01 00 00 00	fixedSizeValue [Int32] 1
02 01 FF 0F	propDef **PidTagEntryId** property (0FFF0102 [Binary])
79 00 00 00	length 121 (0x79)
00 00 00 00- DC A7 40 C8 C0 42 10 1A- B4 B9 08 00 2B 2F E1 82- 01 00 00 00 00 00 00 00- 2F 4F 3D 46 49 52 53 54- 20 4F 52 47	varSizeValue .....@. .B..... +/..... ..../O=F IRST ORG
... value truncated...	
02 01 0B 30	propDef **PidTagSearchKey** property (300B0102 [Binary])
60 00 00 00	length 96 (0x60)
45 58 3A 2F- 4F 3D 46 49	varSizeValue

Bytes on the wire	Value/description
52 53 54 20- 4F 52 47 41 4E 49 5A 41- 54 49 4F 4E 2F 4F 55 3D- 45 58 43 48 41 4E 47 45- 20 41 44 4D	EX:/O=FI RST ORGA NIZATION /OU=EXCH ANGE ADM
... value truncated...	
1F 00 20 3A	propDef **PidTagTransmittableDisplayName** property (3A20001F [Unicode])
06 00 00 00	length 6 (0x6)
74 00 31 00- 00 00	varSizeValue t.1...
0B 00 0F 0E	propDef **PidTagResponsibility** property (0E0F000B [Bool])
01 00	fixedSizeValue [Bool] **TRUE**
0B 00 40 3A	propDef **PidTagSendRichInfo** property (3A40000B [Bool])
01 00	fixedSizeValue [Bool] **TRUE**
03 00 FD 5F	propDef **PidTagRecipientFlags** property (5FFD0003 [Int32])
01 00 00 00	fixedSizeValue [Int32] 1
02 01 F7 5F	propDef **PidTagRecipientEntryId** property (5FF70102 [Binary])
79 00 00 00	length 121 (0x79)
00 00 00 00- DC A7 40 C8 C0 42 10 1A- B4 B9 08 00 2B 2F E1 82- 01 00 00 00 00 00 00 00- 2F 6F 3D 46 69 72 73 74- 20 4F 72 67	varSizeValue .....@. .B..... +/..... ..../o=F irst Org
... value truncated...	

Bytes on the wire	Value/description
1F 00 FE 39	propDef **PidTagSmtpAddress** property (39FE001F [Unicode])
46 00 00 00	length 70 (0x46)
74 00 31 00- 40 00 65 00 75 00 6D 00- 61 00 72 00 75 00 2D 00- 64 00 6F 00 6D 00 2E 00- 65 00 78 00 74 00 65 00- 73 00 74 00	varSizeValue t.1.@.e. u.m.a.r. u.-.d.o. m...e.x. t.e.s.t.
... value truncated...	
03 00 05 39	propDef **PidTagDisplayTypeEx** property (39050003 [Int32])
00 00 00 40	fixedSizeValue [Int32] 1073741824
03 00 00 39	propDef **PidTagDisplayType** property (39000003 [Int32])
00 00 00 00	fixedSizeValue [Int32] 0
03 00 FE 0F	propDef **PidTagObjectType** property (0FFE0003 [Int32])
06 00 00 00	fixedSizeValue [Int32] 6
1F 00 FF 39	propDef **PidTagAddressBookDisplayNamePrintable** property (39FF001F [Unicode])
06 00 00 00	length 6 (0x6)
74 00 31 00- 00 00	varSizeValue t.1...
1F 00 00 3A	propDef **PidTagAccount** property (3A00001F [Unicode])
06 00 00 00	length 6 (0x6)
74 00 31 00- 00 00	varSizeValue t.1...
03 00 DE 5F	propDef

Bytes on the wire	Value/description
	Unspecified property (5FDE0003 [Int32])
00 00 00 00	fixedSizeValue [Int32] 0
03 00 DF 5F	propDef **PidTagRecipientOrder** property (5FDF0003 [Int32])
00 00 00 00	fixedSizeValue [Int32] 0
1F 00 F6 5F	propDef **PidTagRecipientDisplayName** property (5FF6001F [Unicode])
06 00 00 00	length 6 (0x6)
74 00 31 00-00 00	varSizeValue t.1...
03 00 FF 5F	propDef **PidTagRecipientTrackStatus** property (5FFF0003 [Int32])
00 00 00 00	fixedSizeValue [Int32] 0
03 00 04 40	**marker** **EndToRecip** marker (section 2.2.4.1.4) (40040003 [Int32])
03 00 02 40	**marker** **EndEmbed** marker (section 2.2.4.1.4) (40020003 [Int32])
03 00 0E 40	**marker** **EndAttach** marker (section 2.2.4.1.4) (400E0003 [Int32])
03 00 13 40	**marker** **IncrSyncDel** marker (section 2.2.4.1.4) (40130003 [Int32])
02 01 E5 67	propDef **MetaTagIdsetDeleted** property (section 2.2.1.3.1) (67E50102 [Binary])
0D 00 00 00	length 13 (0xD)
01 00 06 00-00 00 78 2E 23 00 04 00-00	varSizeValue ......x. #....
03 00 2F 40	**marker** **IncrSyncRead** marker (section 2.2.4.1.4) (402F0003 [Int32])
02 01 2D 40	propDef **MetaTagIdsetRead** property (section 2.2.1.3.4) (402D0102 [Binary])

Bytes on the wire	Value/description
0A 00 00 00	length 10 (0xA)
01 00 06 00-00 00 78 2E 1F 00	varSizeValue .....x. ..
02 01 2E 40	propDef **MetaTagIdsetUnread** property (section 2.2.1.3.5) (402E0102 [Binary])
0A 00 00 00	length 10 (0xA)
01 00 06 00-00 00 78 2E 20 00	varSizeValue .....x. .
03 00 3A 40	**marker** **IncrSyncStateBegin** marker (section 2.2.4.1.4) (403A0003 [Int32])
02 01 96 67	propDef **MetaTagCnsetSeen** property (section 2.2.1.1.2) (67960102 [Binary])
1D 00 00 00	length 29 (0x1D)
19 D7 FB 0F-06 16 A1 41 BF F6 91 C7-63 DA A8 66 03 00 00 00-52 00 00 01 78 4D 1D 50-00	IDSET printout: {0ffbd719-1606-41a1-bff6-91c763daa866:{[0x1, 0x784D1D]}}
02 01 DA 67	propDef **MetaTagCnsetSeenFAI** (section 2.2.1.1.3) (67DA0102 [Binary])
1D 00 00 00	length 29 (0x1D)
19 D7 FB 0F-06 16 A1 41 BF F6 91 C7-63 DA A8 66 03 00 00 00-52 00 00 01 78 4D 1D 50-00	IDSET printout: {0ffbd719-1606-41a1-bff6-91c763daa866:{[0x1, 0x784D1D]}}
03 00 17 40	propDef **MetaTagIdsetGiven** property (section 2.2.1.1.1) (40170003 [Int32])
38 00 00 00	length 56 (0x38)

Bytes on the wire	Value/description
19 D7 FB 0F-06 16 A1 41 BF F6 91 C7-63 DA A8 66 05 00 00 00-78 2E 52 1D 22 50 00 D2-0C 67 79 AC 4C 50 42 89-2C 24 5D 2D 1A E3 A4 05-00 00 00 78 06 42 01 01-01 0C 50 00	IDSET printout: {0ffbd719-1606-41a1-bff6-91c763daa866:{[0x782E1D, 0x782E22]},79670cd2-4cac-4250-892c-245d2d1ae3a4:{[0x780601, 0x780602], [0x78060C, 0x78060C]}}
02 01 D2 67	propDef **MetaTagCnsetRead** property (section 2.2.1.1.4) (67D20102 [Binary])
1D 00 00 00	length 29 (0x1D)
19 D7 FB 0F-06 16 A1 41 BF F6 91 C7-63 DA A8 66 03 00 00 00-52 00 00 01 78 4D 1D 50-00	IDSET printout: {0ffbd719-1606-41a1-bff6-91c763daa866:{[0x1, 0x784D1D]}}
03 00 3B 40	**marker** **IncrSyncStateEnd** marker (section 2.2.4.1.4) (403B0003 [Int32])
03 00 14 40	**marker** **IncrSyncEnd** marker (section 2.2.4.1.4) (40140003 [Int32])
	EOS

4.6 Conflict Detection and Conflict Resolution Examples

4.6.1 Comparing the PidTagPredecessorChangeList Property to Detect Conficts, No Conflicts Found

In this example, data is being replicated from the client to the server. The **PidTagPredecessorChangeList** properties (section 2.2.1.2.8) of both the client and the server contain one **PidTagChangeKey** property (section 2.2.1.2.7). The value of the **GLOBCNT** structure, as described in section 2.2.2.5, of the client's **PidTagPredecessorChangeList** property entry is larger than the value of the **GLOBCNT** structure that is located on the server.

Before synchronization:

- The client **PidTagPredecessorChangeList** property has the following data:

Size	GUID	GLOBCNT
22	75DCB0E0-EDB1-481E-B5CE-EC3400896353	00 8e 7a 74 08 0a

- The server **PidTagPredecessorChangeList** property has the following data:

Size	GUID	GLOBCNT
22	75DCB0E0-EDB1-481E-B5CE-EC3400896353	00 8e 7a 74 08 08

Conflict analysis:

- There are no common **PidTagChangeKey** property items in the client and server **PidTagPredecessorChangeList** property that are to be eliminated.
- The client **PidTagChangeKey** property is larger than the server **PidTagChangeKey** property, so the imported item from the client overwrites the server item.
- There is no change on the client **PidTagPredecessorChangeList** property.

After synchronization:

- The client **PidTagPredecessorChangeList** property has no change. The value is the same as before synchronization.
- The server **PidTagPredecessorChangeList** property has the following data:

Size	GUID	GLOBCNT
22	75DCB0E0-EDB1-481E-B5CE-EC3400896353	00 8e 7a 74 08 0a

In the next example, data is being replicated from the client to the server, where the client predecessor change list (**PidTagPredecessorChangeList** property) shows that the client has more recent changes than the server.

Before synchronization:

- The client **PidTagPredecessorChangeList** property has the following data:

Size	GUID	GLOBCNT
22	75DCB0E0-EDB1-481E-B5CE-EC3400896353	00 8e 7a 74 08 0a
22	2A47B01B-29A5-45F1-9FDC-F6E14FB7ECCA	00 8e 7a 7c 13 30

- The server **PidTagPredecessorChangeList** property has the following data:

Size	GUID	GLOBCNT
22	75DCB0E0-EDB1-481E-B5CE-EC3400896353	00 8e 7a 74 08 0a

Conflict analysis:

- As stated in the previous conflict-detection logic, removing the identical **PidTagChangeKey** property values from both the **PidTagPredecessorChangeList** properties leaves only the latest **PidTagChangeKey** property of the client (as shown in the following table).

Size	GUID	GLOBCNT
22	2A47B01B-29A5-45F1-9FDC-F6E14FB7ECCA	00 8e 7a 7c 13 30

- This imported client item overwrites the server item.

After synchronization:

- The client **PidTagPredecessorChangeList** property has no change. The value is the same as before synchronization.
- The server **PidTagPredecessorChangeList** property has the following data:

Size	GUID	GLOBCNT
22	75DCB0E0-EDB1-481E-B5CE-EC3400896353	00 8e 7a 74 08 0a
22	2A47B01B-29A5-45F1-9FDC-F6E14FB7ECCA	00 8e 7a 7c 13 30

4.6.2 Comparing the PidTagPredecessorChangeList Property to Detect Conflicts, Conflicts Found

In this example, the data is being replicated from the client to the server with the predecessor change list (**PidTagPredecessorChangeList** property (section 2.2.1.2.8)) so that the client has the same lineage but a new **GLOBCNT** structure value, as described in section 2.2.2.5. The server also has a different **GLOBCNT** structure value.

Before synchronization:

- The client **PidTagPredecessorChangeList** property has the following data:

Size	GUID	GLOBCNT
22	75DCB0E0-EDB1-481E-B5CE-EC3400896353	00 8e 7a 74 08 0a
22	2A47B01B-29A5-45F1-9FDC-F6E14FB7ECCA	00 8e 7a 7c 13 30

- The server **PidTagPredecessorChangeList** property has the following data:

Size	GUID	GLOBCNT
22	75DCB0E0-EDB1-481E-B5CE-EC3400896353	00 8e 7a 74 08 0a
22	0EFAF908-FB24-0EFA-3820-570048EED320	00 8e 7a 7c 3e 5e

Conflict analysis:

- As stated in the previous conflict-detection logic, removing the identical **PidTagChangeKey** property (section 2.2.1.2.7) values from both the client and server **PidTagPredecessorChangeList** properties leaves the following client **PidTagPredecessorChangeList** property:

Size	GUID	GLOBCNT
22	2A47B01B-29A5-45F1-9FDC-F6E14FB7ECCA	00 8e 7a 7c 13 30

and the following server **PidTagPredecessorChangeList** property:

Size	GUID	GLOBCNT
22	0EFAF908-FB24-0EFA-3820-570048EED320	00 8e 7a 7c 3e 5e

- There is a conflict because the server **PidTagPredecessorChangeList** property does not contain the remaining client **PidTagChangeKey** property item, and the client **PidTagPredecessorChangeList** property does not contain the remaining server **PidTagChangeKey** property item.

After synchronization:

- The server creates a conflict message that contains all four items, as shown in the following table.

Size	GUID	GLOBCNT
22	75DCB0E0-EDB1-481E-B5CE-EC3400896353	00 8e 7a 74 08 0a
22	0EFAF908-FB24-0EFA-3820-570048EED320	00 8e 7a 7c 3e 5e
22	75DCB0E0-EDB1-481E-B5CE-EC3400896353	00 8e 7a 74 08 0a
22	2A47B01B-29A5-45F1-9FDC-F6E14FB7ECCA	00 8e 7a 7c 13 30

In the next example, the data is being replicated from the client to the server and the client **PidTagPredecessorChangeList** property has different **PidTagChangeKey** properties than the server.

Before synchronization:

- The client **PidTagPredecessorChangeList** property has the following data:

Size	GUID	GLOBCNT
22	75DCB0E0-EDB1-481E-B5CE-EC3400896353	00 8e 7a 74 08 0a
22	2A47B01B-29A5-45F1-9FDC-F6E14FB7ECCA	00 8e 7a 7c 13 30

- The server **PidTagPredecessorChangeList** property has the following data:

Size	GUID	GLOBCNT
22	75DCB0E0-EDB1-481E-B5CE-EC3400896353	00 8e 7a 74 08 ef

Conflict analysis:

- There are no identical **PidTagChangeKey** properties.
- There is a conflict because the client **PidTagPredecessorChangeList** property does not contain the server **PidTagChangeKey** property item.

After synchronization:

- The server creates a conflict message that contains the three items, as shown in the following table.

Size	GUID	GLOBCNT
22	75DCB0E0-EDB1-481E-B5CE-EC3400896353	00 8e 7a 74 08 ef
22	75DCB0E0-EDB1-481E-B5CE-EC3400896353	00 8e 7a 74 08 0a
22	2A47B01B-29A5-45F1-9FDC-F6E14FB7ECCA	00 8e 7a 7c 13 30

5 Security

5.1 Security Considerations for Implementers

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

5.2 Index of Security Parameters

None.

6 Appendix A: Product Behavior

The information in this specification is applicable to the following Microsoft products or supplemental software. References to product versions include released service packs.

- Microsoft Exchange Server 2003
- Microsoft Exchange Server 2007
- Microsoft Exchange Server 2010
- Microsoft Exchange Server 2013
- Microsoft Exchange Server 2016
- Microsoft Office Outlook 2003
- Microsoft Office Outlook 2007
- Microsoft Outlook 2010
- Microsoft Outlook 2013
- Microsoft Outlook 2016

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

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

<1> Section 2.2.1.3.3: The **MetaTagIdsetExpired** property is not supported in Exchange 2013 and Exchange 2016.

<2> Section 2.2.3: In Exchange 2003 and Exchange 2007, the **RopSynchronizationGetTransferState** ([MS-OXCROPS] section 2.2.13.8) and the **RopFastTransferSourceGetBuffer** ([MS-OXCROPS] section 2.2.12.3) ROPs are used to checkpoint ICS download operations. In Exchange 2010, Exchange 2013, and Exchange 2016, trying to retrieve a checkpoint ICS state during the download returns the initial state, and not a state with differences applied.

<3> Section 2.2.3.1.1.1.1: The **BestBody** flag is not supported in Exchange 2013 and Exchange 2016. In Exchange 2013 and Exchange 2016, the message body is always in the original format.

<4> Section 2.2.3.1.1.1.1: Exchange 2003 and Office Outlook 2003 do not support partial item downloads. Exchange 2003 does not support the **PartialItem** bit flag of the **SendOptions** field and Office Outlook 2003 does not pass it.

<5> Section 2.2.3.1.1.3.1: The **BestBody** flag is not supported in Exchange 2013 and Exchange 2016. In Exchange 2013 and Exchange 2016, the message body is always in the original format.

<6> Section 2.2.3.1.1.4.1: In Exchange 2003 and Exchange 2007, the **Move** bit flag is not ignored on receipt.

<7> Section 2.2.3.1.1.4.1: In Exchange 2003 and Exchange 2007, the **Move** bit flag is read by the server.

<8> Section 2.2.3.1.1.5.2: The **NoRoom** value is not returned by Exchange 2010, Exchange 2013, or Exchange 2016. The **NoRoom** value is supported by Exchange 2003 and Exchange 2007.

<9> Section 2.2.3.1.1.5.2: Exchange 2010, Exchange 2013, and Exchange 2016 report inaccurate information in this output parameter when client connection services are deployed on an Microsoft Exchange Server server that does not also have a mailbox message store installed.

<10> Section 2.2.3.1.2.2.1: Exchange 2003, Exchange 2007, Exchange 2010, the initial release of Exchange 2013, Office Outlook 2003, Office Outlook 2007, Outlook 2010, and the initial release of Outlook 2013 do not support the **Execute** request type. The **Execute** request type was introduced in Microsoft Outlook 2013 Service Pack 1 (SP1) and Microsoft Exchange Server 2013 Service Pack 1 (SP1).

<11> Section 2.2.3.1.2.2.2: Exchange 2010, Exchange 2013, and Exchange 2016 set the value of the **TotalStepCount** field to 0x0001. Exchange 2003 and Exchange 2007 set the value of the **TotalStepCount** field to 0x0000.

<12> Section 2.2.3.1.2.2.2: Exchange 2003, Exchange 2007, Exchange 2010, Exchange 2013, and Exchange 2016 always return a value for the **BufferSizeUsed** field that is equal to the value of the **TransferDataSize** field, regardless of whether the value of the **ReturnValue** field is **Success** (0x00000000).

<13> Section 2.2.3.2.1.1.1: The **BestBody** flag is not supported in Exchange 2013 and Exchange 2016. In Exchange 2013 and Exchange 2016, the message body is always in the original format.

<14> Section 2.2.3.2.4.2.1: The **FailOnConflict** flag is not supported by Exchange 2003, Exchange 2007, or the initial release version of Exchange 2010. The **FailOnConflict** flag is supported by Microsoft Exchange Server 2010 Service Pack 1 (SP1), Exchange 2013, and Exchange 2016.

<15> Section 2.2.3.2.4.3.1: The initial release version of Exchange 2010 has a return value of InvalidParameter (0x80070057) if the value of the **PidTagParentSourceKey** property (section 2.2.1.2.6) has a length of zero.

<16> Section 2.2.3.2.4.5.1: The **HardDelete** flag is not supported by Exchange 2003 or Exchange 2007.

<17> Section 2.2.4.1.3: Exchange 2003 and Exchange 2007 fail to add a null-terminator when string values are larger than 32 KB.

<18> Section 2.2.4.3.5: Exchange 2003, Exchange 2007, and Exchange 2010 do not include the **PidTagParentFolderId** property (section 2.2.1.2.4) in the **folderChange** element if the **Eid** flag of the **SynchronizationExtraFlags** field is set.

<19> Section 2.2.4.3.6: Exchange 2010, Exchange 2013, and Exchange 2016 do not include the **MetaTagEcWarning** meta-property (section 2.2.4.1.5.2) in the **propList** element because Exchange 2010, Exchange 2013, and Exchange 2016 do not check permissions on move operations.

<20> Section 2.2.4.3.15: Exchange 2003 and Office Outlook 2003 do not support partial item downloads. Exchange 2003 does not support the **PartialItem** flag of the **SendOptions** field and Office Outlook 2003 does not pass it.

<21> Section 2.2.4.3.20: Exchange 2003 and Exchange 2007 can include the **MetaTagEcWarning** meta-property (section 2.2.4.1.5.2) in the **propList** of the **folderContent** element, as described in section 2.2.4.3.6.

<22> Section 2.2.4.4: Office Outlook 2003, Office Outlook 2007, Outlook 2010, Outlook 2013, and Outlook 2016 do not use the **folderContent** element as a root element in a FastTransfer stream.

<23> Section 2.2.4.4: The **RopFastTransferSourceCopyProperties** ROP (section 2.2.3.1.1.2) does not use the **attachmentContent** element as a root element in a FastTransfer stream in Office Outlook 2003, Office Outlook 2007, Outlook 2010, Outlook 2013, or Outlook 2016.

<24> Section 3.1.1: Exchange 2003, Exchange 2007, Exchange 2010, and the initial release of Exchange 2013 do not support the **session context cookie**. The **session context cookie** was introduced in Exchange 2013 SP1.

<25> Section 3.1.5.4.3.2: Exchange 2010, Exchange 2013, and Exchange 2016 send an **RpcFormat** error (0x000004B6), and Exchange 2003 and Exchange 2007 send a **FormatError** error (0x000004ED), as specified in [MS-OXCDATA] section 2.4.1, if the server encounters an unsupported command or any other decoding failures.

<26> Section 3.1.5.4.3.2.3: Exchange 2010, Exchange 2013, and Exchange 2016 send an **RpcFormat** error (0x000004B6), and Exchange 2003 and Exchange 2007 send a **FormatError** error (0x000004ED), as specified in [MS-OXCDATA] section 2.4.1, if the server encounters the **Bitmask** command when there are more or fewer than five bytes in the common byte stack.

<27> Section 3.1.5.4.3.2.4: Exchange 2010, Exchange 2013, and Exchange 2016 send an **RpcFormat** error (0x000004B6) and Exchange 2003 and Exchange 2007 send a **FormatError** error (0x000004ED), as specified in [MS-OXCDATA] section 2.4.1, if the high value of the range is larger than the low value of the range.

<28> Section 3.2.1: Exchange 2003, Exchange 2007, Exchange 2010, and the initial release of Exchange 2013 do not support the **session context cookie**. The **session context cookie** was introduced in Exchange 2013 SP1.

<29> Section 3.2.5.3: Exchange 2010, Exchange 2013, and Exchange 2016 do not perform this confirmation step. Exchange 2003 and Exchange 2007 do perform this confirmation step.

<30> Section 3.2.5.7: Exchange 2003 and Office Outlook 2003 do not support partial item downloads. Exchange 2003 does not support the **PartialItem** bit flag of the **SendOptions** field and Office Outlook 2003 does not pass it.

<31> Section 3.2.5.8.1.1: Exchange 2010, Exchange 2013, and Exchange 2016 do not support the **Move** flag for the **RopFastTransferSourceCopyTo** ROP (section 2.2.3.1.1.1). The server sets the value of the **ReturnValue** field to **InvalidParameter** (0x80070057) if it receives this flag.

<32> Section 3.2.5.8.1.1: The **BestBody** flag is not supported in Exchange 2013 and Exchange 2016. In Exchange 2013 and Exchange 2016, the message body is always in the original format.

<33> Section 3.2.5.8.1.3: The **BestBody** flag is not supported in Exchange 2013 and Exchange 2016. In Exchange 2013 and Exchange 2016, the message body is always in the original format.

<34> Section 3.2.5.8.1.5: Exchange 2003, Exchange 2007, Exchange 2010, the initial release of Exchange 2013, Office Outlook 2003, Office Outlook 2007, Outlook 2010, and the initial release of Outlook 2013 do not support the **Execute** request type. The **Execute** request type was introduced in Outlook 2013 SP1 and Exchange 2013 SP1.

<35> Section 3.2.5.8.1.5: Exchange 2003, Exchange 2007, Exchange 2010, the initial release of Exchange 2013, Office Outlook 2003, Office Outlook 2007, Outlook 2010, and the initial release of Outlook 2013 do not support the **Execute** request type. The **Execute** request type was introduced in Outlook 2013 SP1 and Exchange 2013 SP1.

<36> Section 3.2.5.8.2.1: Exchange 2010, Exchange 2013, and Exchange 2016 fail the ROP if unknown bit flags in the **CopyFlags** field are set.

<37> Section 3.2.5.8.2.1: Exchange 2003 and Exchange 2007 ignore unknown values of the **CopyFlags** field.

<38> Section 3.2.5.9.1.1: The **BestBody** flag is not supported in Exchange 2013 and Exchange 2016. In Exchange 2013 and Exchange 2016, the message body is always in the original format.

<39> Section 3.2.5.9.3.1: In Exchange 2010, Exchange 2013, and Exchange 2016, the **Error! Hyperlink reference not valid.** ROP (section 2.2.3.2.3.1) returns the initial ICS state for the download context until the end of the FastTransfer stream has been downloaded. Exchange 2003 and Exchange 2007 do not use this behavior.

<40> Section 3.2.5.9.3.1: In Exchange 2003 and Exchange 2007, the **Error! Hyperlink reference not valid.** ROP (section 2.2.3.2.3.1) returns a checkpoint ICS state that is reflective of the current status. Exchange 2010, Exchange 2013, and Exchange 2016 do not use this behavior.

<41> Section 3.2.5.9.4.2: Exchange 2010, Exchange 2013, and Exchange 2016 fail the ROP if unknown bit flags are set. Exchange 2003 and Exchange 2007 do not fail the ROP if unknown bit flags are set.

<42> Section 3.3.1: Exchange 2003, Exchange 2007, Exchange 2010, and the initial release of Exchange 2013 do not support the **session context cookie**. The **session context cookie** was introduced in Exchange 2013 SP1.

<43> Section 3.3.5.1: Office Outlook 2003 introduced the implementation of performing upload operations before download operations; however the Bulk Data Transfer Protocol has no requirement about the order of upload or download operations.

<44> Section 3.3.5.6: Exchange 2003 and Exchange 2007 do not return the initial ICS state in place of the checkpoint ICS state.

<45> Section 3.3.5.6: In Exchange 2003 and Exchange 2007, synchronization download operations function the same as synchronization upload operations. In Exchange 2003 and Exchange 2007, the server returns checkpoint ICS states that are accurate to the time at which the checkpoint was requested in both synchronization download operations and synchronization upload operations.

<46> Section 3.3.5.7.1: In Exchange 2003 and Exchange 2007, clients are not required to pass a value in the **BufferSize** field of a certain size.

<47> Section 3.3.5.9.1: Office Outlook 2003 does not support the 0x00000480 error code.

7 Change Tracking

This section identifies changes that were made to this document since the last release. Changes are classified as New, Major, Minor, Editorial, or No change.

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

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

- A document revision that incorporates changes to interoperability requirements or functionality.
- The removal of a document from the documentation set.

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

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

The revision class **No change** means that no new technical changes were introduced. Minor editorial and formatting changes may have been made, but the technical content of the document is identical to the last released version.

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

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

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

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

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

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

Section	Tracking number (if applicable) and description	Major change (Y or N)	Change type
4.5 FastTransfer Stream Produced by a Content Synchronization Download Example	Updated the table showing bytes on the wire.	N	Content update.
7 Change Tracking	Updated the list of products to which product behavior notes throughout the document apply.	N	Product behavior note updated.

8 Index