[MS-OXCFXICS]: Bulk Data Transfer Protocol Specification

Intellectual Property Rights Notice for Protocol Documentation

- **Copyrights.** This protocol documentation is covered by Microsoft copyrights. Regardless of any other terms that are contained in the terms of use for the Microsoft website that hosts this documentation, you may make copies of it in order to develop implementations of the protocols, and may distribute portions of it in your implementations of the protocols or your documentation as necessary to properly document the implementation. 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 protocol documentation.
- **No Trade Secrets.** Microsoft does not claim any trade secret rights in this documentation.
- **Patents.** Microsoft has patents that may cover your implementations of the protocols. Neither this notice nor Microsoft's delivery of the documentation grants any licenses under those or any other Microsoft patents. However, the protocols may be covered by Microsoft's Open Specification Promise (available here: http://www.microsoft.com/interop/osp). If you would prefer a written license, or if the protocols are not covered by the OSP, patent licenses are available by contacting protocol@microsoft.com.
- **Trademarks.** The names of companies and products contained in this documentation may be covered by trademarks or similar intellectual property rights. This notice does not grant any licenses under those rights.

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

Tools. This protocol documentation is 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. A protocol specification does 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.

Revision Summary			
Author	Date	Version	Comments
Microsoft Corporation	April 4, 2008	0.1	Initial Availability.
Microsoft Corporation	April 25, 2008	0.2	Revised and updated property names and other technical content.
Microsoft Corporation	June 27, 2008	1.0	Initial Release.

Microsoft Corporation	August 6, 2008	1.01	Revised and edited technical content.
Microsoft Corporation	September 3, 2008	1.02	Revised and edited technical content.
Microsoft Corporation	December 3, 2008	1.03	Revised and edited technical content.
Microsoft Corporation	February 4, 2009	1.04	Revised and edited technical content.
Microsoft Corporation	March 4, 2009	1.05	Editorial updates.

Table of Contents

1 Introduction

This document specifies a protocol for 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 document specifies the following:

- How a client can configure a **remote operation (ROP)** to download or **upload** a set of folders or messages to or 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.

1.1 Glossary

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

ABNF
attachment
Attachment object
change number (CN)
Embedded Message object
enterprise/site/server distinguished name (ESSDN)
folder
folder associated information (FAI)
folder ID (FID)
Folder object
ghosted folder
global counter (GLOBCNT)
global identifier (GID)
GLOBSET
GUID
handle
ICS state
Incremental Change Synchronization (ICS)
little-endian
local replica
LongTermID
mailbox
message
message ID (MID)

Message object
messaging object
Predecessor Change List (PCL)
property
property tag
property type
public folder
recipient
remote operation (ROP)
remote procedure call (RPC)
replica GUID (REPLGUID)
replica ID (REPLID)
restriction
Rich Text Format (RTF)
ROP request buffer
ROP response buffer
server replica
ShortTermIDstore
subobject
synchronization download context
synchronization scope
synchronization upload context
Unicode

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

BOOLEAN
BYTE

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

PtypBinary
PtypBoolean
PtypErrorCode
PtypGuid
PtypInteger16
PtypInteger32
PtypInteger64
PtypServerId
PtypString
PtypString8

The following terms are specific to this document:

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.

change number set (CNSET): A data structure that is similar to an **IDSET**, in which the **GLOBCNTs** represent changes rather than **messaging objects**.

checkpoint ICS state: The **ICS state** provided by the server in the middle of an **ICS** operation, which reflects the state of the **local replica**, indicated by **initial ICS state**, after applying all differences transmitted in the **ICS** operation.

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. Used in a last-in first-out (LIFO) fashion during serialization or deserialization of **GLOBSETs**, as specified in section 2.2.2.4.1.

conflict detection: The process used to detect that two versions of the same object are in conflict with each other, that is, one is not a direct or an indirect predecessor of another.

conflict handling: Actions taken upon detection of a conflict between versions of an object. Includes **conflict detection**, **conflict reporting**, and **conflict resolution**.

conflict reporting: The automated process of notifying a system actor of a previously detected conflict.

conflict resolution: The automated or semi-automated process of resolving a previously detected conflict between versions of an object by replacing conflicting versions with their successor. How the successor version is related to the conflicting version depends on the **conflict resolution** algorithm used.

contents 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.

download: Transmission of data (payload) from a server to a client.

embedded message: See **Embedded Message objects**.

expired Message object: A **Message object** that the server has removed due to its age.

external identifier (XID): A globally unique identifier for an entity that represents either a **foreign identifier** or an **internal identifier**. 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 a XID represents and internal identifier, then it can be also called a **GID**.

FastTransfer stream: A binary format for encoding full or partial **folder** and **message** data. Also encodes information about differences between **mailbox replicas**.

final ICS state: The **ICS state** provided by the server upon completion of an **ICS** operation. **Final ICS state** is a **checkpoint ICS state** provided at the end of the **ICS** operation.

foreign identifier: An identifier of an entity assigned by a foreign system, usually a client. Always has a form of an **XID**, but not all **XIDs** are **foreign identifiers**.

formatted IDSET: An IDSET that has been properly arranged for serialization in a series of **REPLID**-constant sections that are sorted by **REPLID** in ascending order. Each section is a **GLOBSET**. This logical representation is further compressed on the wire.

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

IDSET: A set of IDs, or **REPLID** and **GLOBCNT** pairs. An **IDSET** has to be represented as a **formatted IDSET** to be serialized on the wire.

IFF: Logical equivalence, that is A **IFF** B is the same as "A if and only if B".

initial ICS state: The **ICS state** that is provided by the client when it configures an **ICS** operation.

Input Server object: An object on a server that is used as input for remote operations. For more details about Server objects, see [MS-OXCROPS] section 3.

internal identifier: An identifier of a **mailbox** entity assigned by a server, which corresponds to a format and **restrictions** specified in [MS-OXCSTOR].
Short-term representations of **internal identifiers**, which consist of a 2-byte **REPLID** and a 6-byte **GLOBCNT,** are scoped to the logon in which they were obtained.
If the term *internal identifier* is mentioned on its own, it means a **short-term** representation of such. See also: **GID**.

marker: Unsigned 32-bit integer values, which adhere to **property tag** syntax and are used to denote the start and end of related data in **fast transfer streams**. **Property tags** that are used by **markers** do not represent valid properties. For a full list of **markers**, see section 2.2.4.1.4.

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

normal message: Any **message** that is not an **FAI message**.

Output Server object: An object on a server that is used as an output for remote operations. For more details about Server objects, see [MS-OXCROPS] section 3.

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

property list restriction table: A set of **restrictions** imposed on an array of **properties** and their values, expressed in the tabular form specified in section 2.2.

top-level message: A **message** that is not an **embedded message**. **Top-level message**s are **messaging objects**.

upload: Transmission of data (payload) from a client to a server.

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

1.2 References

1.2.1 Normative References

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

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

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

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

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

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

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

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

[MS-OXCSYNC] Microsoft Corporation, "Mailbox Synchronization Protocol Specification", June 2008.

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

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

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

[RFC4234] Crocker, D., Ed. and Overell, P., "Augmented BNF for Syntax Specifications: ABNF", RFC 4234, October 2005, http://www.ietf.org/rfc/rfc4234.txt.

1.2.2 Informative References

None.

1.3 Protocol Overview

This document specifies 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 **remote operation** (**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 document supports the transfer of data in scenarios that derive from the following semi-indepenent variables:

1. Direction of data transmission: **download** or **upload**.
2. Type of **messaging object**s 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 **FIDs** and **MIDs**
    - 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.

1.3.1 Fast Transfer Copy Operations

FastTransfer enables clients to efficiently copy the content of explicitly specified **folders**, **messages**, and **attachments** between replicas 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. 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 upload operations enable a client to create new folders or modify content of existing folders, messages, and attachments by using input data encoded in the FastTransfer stream format.

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 synchronizations**.

Changes to message properties, changes to read and unread message state, changes to **recipient** and **attachment** information, message creations, and message deletions are included in **contents synchronizations**.

Hierarchy synchronizations and contents synchronizations are the actual processes used to implement ICS on the client and server.

ICS can also be used to send notifications to servers and clients. For details about ICS notifications, see [MS-OXCNOTIF].

1.3.2.1 Download

Information about all changes and deletions made to **mailbox** data on the server is **downloaded** to the client through one or more iterations of a single **ROP**, whose response buffer can be efficiently packed at the **RPC** level.

Performing a **hierarchy synchronization** download using a synchronization context that was opened on a **folder** will produce 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 **contents synchronization** download using a synchronization context that was opened on a folder will produce information about all **message** changes and message deletions in the folder that have happened since the last synchronization download, as defined by the initial ICS state.

1.3.2.2 Upload

Uploading **mailbox** changes from a client to a server resembles the **ICS download** process, except that instead of streaming data through a single **ROP**, multiple individual ROPs are sent to upload changes to individual objects within a mailbox.

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

The Mailbox Synchronization Protocol Specification describes how to apply this protocol to the replication of **mailbox** data between clients and servers, as specified in [MS-OXCSYNC].

The Core Notifications Protocol Specification describes **ICS** notifications, as specified in [MS-OXCNOTIF].

This specification relies on the following:

- An understanding of remote procedure calls (RPCs) and **remote operations (ROPs)**, as specified in [MS-OXCRPC] and [MS-OXCROPS] respectively.

- An understanding of **folders** and **messages**, as specified in [MS-OXCFOLD] and [MS-OXCMSG] respectively.

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 details 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, as specified in [MS-OXCSYNC].

- 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 mentioned in this section. However, it MAY not be appropriate for use in the following scenarios:

- Those requiring copying of data between **folders** of the same mailbox, or different mailboxes residing on the same server. Consider using **RopCopyTo**, as specified in [MS-OXCROPS] section 2.2.7.12, for maximum efficiency.

- Those requiring fine-grain control over the set of information that has to be transferred for each **message**. Consider using other ROPs specified in [MS-OXCROPS] that provide access to individual parts of messages.

- Those that impose constraints on the amount of data that has to be passed over the wire or stored on the client.

- Those that do not allow for persistence of state information on the client between runs.

1.7 Versioning and Capability Negotiation

Localization: Localization-related aspects of the protocol are specified in section 2.2.3.1.1.2.

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 during the connect phase (by the **EcDoConnect** and **EcDoConnectEx** RPCs) of the **RPC** session. For more details, see [MS-OXCRPC] section 3.1.9.

Client version	Description
11.0.0.4920 and above	The client supports receiving **ServerBusy** in the **ReturnValue** field of the **RopFastTransferSourceGetBuffer** response. For more details, see section 2.2.3.1.1.5.
12.0.3730.0 and above	The client supports send optimization for **ICS** using **PidTagTargetEntryId**. For more details, see [MS-OXCSYNC] section 3.1.5.2.2.1.2.

Server version	Description
8.0.359.0 and above	The server supports **PartialItem SendOptions** flag. For more details, see section 2.2.3.1.1.1.2. Clients MUST NOT pass this flag for earlier server versions.

RopTellVersion is used to explicitly declare capabilities of the servers in the server-to-client-to-server **upload** scenario. For details, see section 3.3.4.1.2.1.

1.8 Vendor-Extensible Fields

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

All undefined bits in flag structures and undefined values of enumerations that are defined in this specification are reserved; clients MUST pass 0.

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 from 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:

PidTagCnset*. Refers to any of the following properties: **PidTagCnsetSeen**, **PidTagCnsetSeenFAI**, and **PidTagCnsetRead**.

RopFastTransferSourceCopy*. Refers to any of the following **ROPs**: **RopFastTransferSourceCopyTo**, **RopFastTransferSourceCopyProperties**, **RopFastTransferSourceCopyMessages**, and **RopFastTransferSourceCopyFolder**.

RopSynchronizationImport*. Refers to any of the following ROPs: **RopSynchronizationImportMessageChange**, **RopSynchronizationImportHierarchyChange**, **RopSynchronizationImportMessageMove**, **RopSynchronizationImportDeletes**, **RopSynchronizationImportReadStateChanges**.

RopSynchronizationUploadStateStream*. Refers to any of the following ROPs: **RopSynchronizationUploadStateStreamBegin**, **RopSynchronizationUploadStateStreamContinue**, and **RopSynchronizationUploadStateStreamEnd**.

Sections 2.2.1 through 2.2.4.4 use **property list restriction tables** in the following format to describe **restrictions** on arrays of **property** values:

Name	Restrictions	Comments
PidSomeProperty	Conditional Fixed position …	Condition of existence.
< other properties >	*Prohibited*	Comments.

Any property MUST NOT exist in a property list restriction table more than once. All non-italicized rows of the table represent a **restriction** that is imposed on the property identified in

the **Name** column. For a list of all possible properties, see [MS-OXCPROPS]. The **Comments** column contains free-form comments that amend the meaning of the **Name** and **Restriction** columns. The **Restrictions** column specifies a subset of the following restrictions:

- **Optional** [default]: The property MAY be present in the list.
- **Required**: The property MUST be present in the list.
- **Fixed position**: The position of the property within the list 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 list is conditional. See the **Comments** column for conditions.

Prohibited: The property MUST NOT be present in the list. 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.

2.2.1 Properties

2.2.1.1 ICS State Properties

ICS uses a set of properties known as the **ICS state** to enable a server to narrow down the set of data passed during an incremental change synchronization. By using the **ICS state**, 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 that need to be downloaded 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**.

All properties specified in this section are part of the ICS state. Two of these properties are used for **hierarchy synchronization**. All four properties are used for contents synchronization. The ICS state determines the state of the **local replica** bounded by the synchronization scope (section 3.3.1.2) specified by the client in the **RopSynchronizationConfigure** request (section 2.2.3.2.1.1).

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 need to be downloaded to the client. For more server-specific details, see section 3.2.4.1. Ordinarily, the server modifies the ICS state properties and sends them back to the client. For details about exceptions and checkpointing, see [MS-OXCSYNC] section 3.1.5.3.9.1.

All ICS state properties are of the **PtypBinary** type, and contain a serialized **IDSET** in the **REPLGUID**-based form (section 2.2.2.3.1).

Note that for the purposes of reducing the wire size of the ICS state by enabling compacting of regions (as specified in section 3.2.1.2) and optimizing for performance of determining a set of differences to be downloaded to clients, servers MAY include extra IDs in IDSETs that represent **change numbers sets** (**CNSETs**), as long as that will never affect the sets of differences that are downloaded to clients. For more server-specific details, see the following **property** comments and section 3.2.4.1.

During the first synchronization of a synchronization scope, a client MUST <1> send the relevant ICS state properties as zero-length byte arrays.

2.2.1.1.1 PidTagIdsetGiven

A **PtypBinary** value that contains a serialized **IDSET** of **folder IDs** (**FIDs**) for **hierarchy synchronization**, or **message IDs** (**MIDs**) for **contents synchronization**, that exist in the **local replica** of the client. This IDSET MUST NOT include any IDs that are not in the local replica of the client. Because of this **restriction** on IDs, this **property** might not compress as well as the **PidTagCnset*** properties, which will make the **PidTagIdsetGiven** property grow much bigger than the **PidTagCnset*** properties. For more details about compression of IDSETs, see section 3.2.1.2.

The **property tag** for this property suggests that it is of type **PtypInteger32**, but the data MUST be handled as **PtypBinary** data.<2>

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 **RopSynchronizationGetTransferState**. Clients SHOULD <3> 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 [MS-OXCSYNC].

2.2.1.1.2 PidTagCnsetSeen

A **PtypBinary** value that contains an **IDSET** of **CNs**. The CNs track changes to **folders** (for **hierarchy sychronizations**) or **normal messages** (for **contents synchronizations**) 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 PidTagCnsetSeenFAI

A **PtypBinary** value, with semantics identical to **PidTagCnsetSeen**, except that it contains IDs for **folder associated information** (**FAI)** messages and is therefore only used in **contents synchronization**.

2.2.1.1.4 PidTagCnsetRead

A **PtypBinary** value that contains an **IDSET** of **CNs**. The CNs 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**.

The read state of a message is determined from the **PidTagMessageFlags** property, which contains a bitmask of flags that indicates the origin and current state of the message. For more details about this **property**, see [MS-OXPROPS] section 2.1667.

2.2.1.2 Messaging Object Identification and Change Tracking Properties

This section contains information about the properties that are used by this protocol to identify **messages** and **folders** and track changes.

For details about how **messaging object** and change identification values are created and modified by the protocol roles, see section 3.1.1.

2.2.1.2.1 PidTagMid

A **PtypInteger64** value that contains the **MID** of the **message** currently being synchronized.

For details about the conditions of its presence in message change headers, see section 2.2.3.2.1.1.3.

2.2.1.2.2 PidTagFolderId

A **PtypInteger64** value that contains the **FID** of the **folder** currently being synchronized.

For details about the conditions of its presence in **message** change headers, see section 2.2.3.2.1.1.3.

2.2.1.2.3 PidTagChangeNumber

A **PtypInteger64** value that contains the **CN** that identifies the last change to the **message** or **folder** that is currently being synchronized.

For details about the conditions of its presence in message change headers, see section 2.2.3.2.1.1.4.

2.2.1.2.4 PidTagParentFolderId

A **PtypInteger64** value that contains the **FID** that identifies the parent **folder** of the **messaging object** being synchronized. If a **hierarchy synchronization** download is occuring, this **property** MUST be set to 0 to identify the child of a folder for which the download operation was configured.

2.2.1.2.5 PidTagSourceKey

A **PtypBinary** value that contains an **internal identifier (GID)** for this **folder** or **message**. The binary content of this **property** is a serialization of an XID. For more details about the binary format, see section 2.2.2.1.

For more details about how clients generate this property, see section 3.3.1.1.1.

When requested by clients, servers MUST output the property value if it is persisted, or generate it on-the-fly if it is missing, based on the **internal identifiers** maintained by the server for the **messaging object**, as specified in section 3.2.4.2. For more details about messaging object identification, see section 3.1.1.1.

2.2.1.2.6 PidTagParentSourceKey

A **PtypBinary** value on a **folder** that contains the **PidTagSourceKey** of the parent folder.

2.2.1.2.7 PidTagChangeKey

A **PtypBinary** value that contains the serialized **XID** of the last change to the **messaging object**.

If the last change to the messaging object was imported from a client by using **RopSynchronizationImportMessageChange**, this **property** contains a value for the **PidTagChangeKey** property that was passed in fields to that **ROP**.

If the last change to a messaging object was made by a server, this property contains an XID generated from the **PidTagChangeNumber** property. For more details about generating XIDs based on internal identifiers, see section 3.2.4.2.

2.2.1.2.8 PidTagPredecessorChangeList

A **PtypBinary** value that contains a serialized representation of a **PredecessorChangeList** structure, as specified in section 2.2.2.2. This value represents a set of **change numbers** for versions of the **messaging object** in all replicas that were integrated into the current version. This **property** is used in **conflict detection** by all protocol roles.

2.2.1.3 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** type, and contain a serialized **IDSET** in the **REPLID**-based form (as specified in section 2.2.2.3.1).

2.2.1.3.1 PidTagIdsetDeleted

A **PtypBinary** value that contains a serialization of a **REPLID**-based **IDSETs**. The IDSETs contains the IDs of **folders** (for **hierarchy synchronization**) or **messages** (for **contents synchronization**) that were hard- or soft-deleted since the last synchronization identified by the **initial ICS state**.

2.2.1.3.2 PidTagIdsetSoftDeleted

A **PtypBinary** value that contains a serialization of a **REPLID**-based **IDSETs**. The IDSETs contains the IDs of **messages** that got out of **synchronization scope** since the last synchronization identified by the **initial ICS state**. Note that soft-deleted messages will be reported in the **PidTagIdsetDeleted** property.

2.2.1.3.3 PidTagIdsetExpired

A **PtypBinary** value that contains a serialization of a **REPLID**-based **IDSETs**. The IDSETs contains IDs of **expired Message objects** in a **public folder** that expired since the last synchronization identified by the **initial ICS state**.

2.2.1.3.4 PidTagIdsetRead

A **PtypBinary** value that contains a serialization of a **REPLID**-based **IDSETs**. The IDSETs contain IDs of **messages** that were marked as read (as specified by the **PidTagMessageStatus** property in [MS-OXPROPS] section 2.1732) since the last synchronization identified by the **initial ICS state**.

2.2.1.3.5 PidTagIdsetUnread

A **PtypBinary** value that contains a serialization of a **REPLID**-based **IDSETs**. The IDSETs contain IDs of **messages** that were marked as unread (as specified by the **PidTagMessageStatus** property in [MS-OXPROPS] section 2.1732) since the last synchronization identified by the **initial ICS state**.

2.2.1.4 PidTagAssociated

A **PtypBoolean** value that specifies whether the **message** being synchronized is an **FAI** message.

2.2.1.5 PidTagMessageSize

An unsigned **PtypInteger32** value that identifies the size of the **message** in bytes.

For details about the conditions of the **PidTagMessageSize** presence in message change headers, see section 2.2.3.2.1.1.3.

A server SHOULD make the best effort to calculate this **property**, but because there is no objective way of computing it, it MUST be treated only as an estimate by client.

2.2.1.6 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, **RopSynchronizationConfigure** and **RopFastTransferSouceCopyTo.**

- As values of **PidTagFXDelProp** meta-properties (as specified in section 2.2.4.1.5.1).

Folder properties	Description
PidTagContainerContents	Identifies all **normal messages** in the current **folder**.
PidTagFolderAssociatedContents	Identifies all **FAI** messages in the current folder.
PidTagContainerHierarchy	Identifies all subfolders of the current folder.

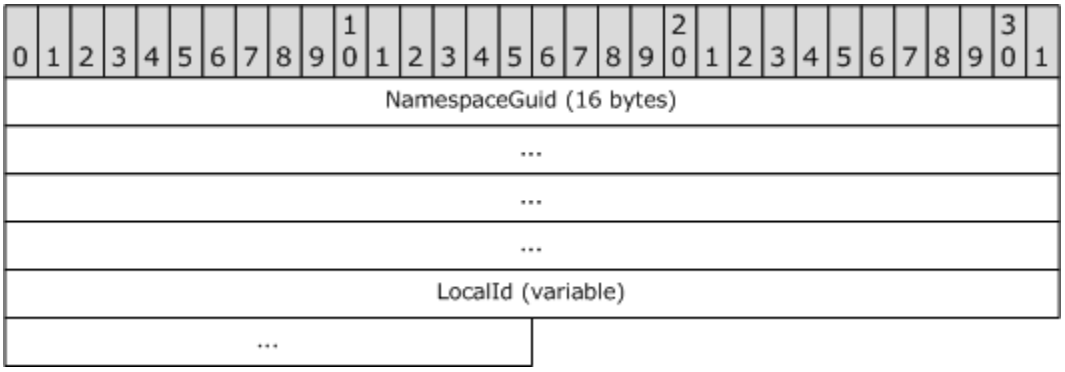
Message Properties	Description
PidTagMessageRecipients	Identifies all **recipients** of the current **message**.
PidTagMessageAttachments	Identifies all **attachments** to the current message.

Attachment Properties	Description
PidTagAttachDataObject	Identifies the **embedded message** of the current attachment.

2.2.2 Structures

2.2.2.1 XID

Represents an **external identifier** for an entity within a data **store**.



NamespaceGuid (16 bytes): A 128-bit **GUID** that identifies the namespace that the identifier specified by **LocalId** belongs to.

LocalId (variable): A variable binary value that contains the ID of the entity in the namespace specified by **NamespaceGuid**. The length of this field MUST be within the [1; 239] range.

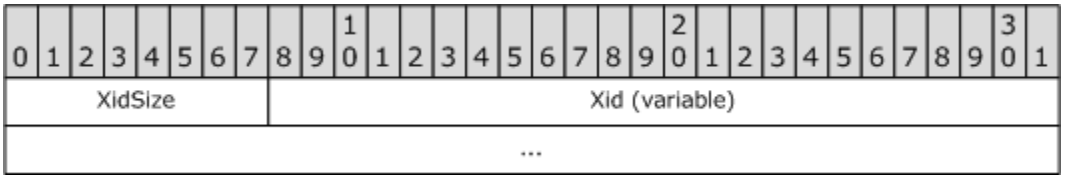
For more details about **GID** structures, which are a subtype of an **XID**, see [MS-OXCSTOR]. For GIDs, the **REPLGUID** maps to the **NamespaceGuid** field, and the global counter (**GLOBCNT)** maps to the **LocalId** field.

All XIDs with the same **NamespaceGuid** MUST have the same length of **LocalId** fields. However, the size of the **LocalId** value cannot be determined by examining the **NamespaceGuid** value and MUST be provided externally. In most cases, XID structures are present within other structures, which specify the size of the XID, such as the **SizedXid** element (as specified in section 2.2.2.2.1) or the **propValue** element (as specified in section 2.2.4.3.21).

2.2.2.2 PredecessorChangeList

Contains a set of **XIDs** that represent **change numbers** of **messaging objects** in different replicas. 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.

2.2.2.2.1 SizedXid



XidSize (1 byte): An unsigned 8-bit integer. MUST be equal to the size of the **Xid** field in bytes.

Xid (variable): A structure of type **Xid** that contains the value of the **internal identifier** of an object, or internal or **external identifier** of a **change number**. This field MUST contain the same number of bytes as specified in the **XidSize** field.

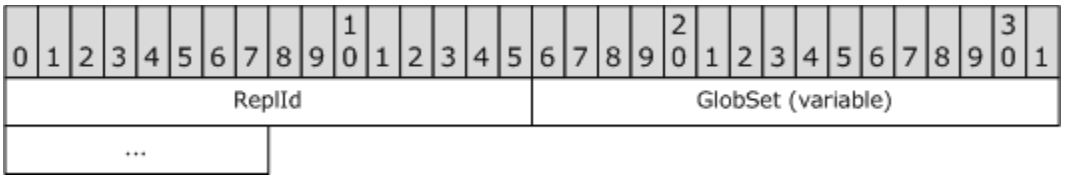
2.2.2.3 IDSET

An **IDSET** is a set of ID values. The IDSET can be used to contain a set of **MID** values, a set of **FID** values, or a set of **CN** values (also known as **CNSET**). An IDSET MUST NOT contain duplicate ID values.

The serialization format specified in the following sections is optimized for data transfer, and is not intended for in-memory operations. See section 3.1.1.3 for details about the serialization and deserialization process.

2.2.2.3.1 Serialized IDSET with REPLID

For every **REPLID** and **GLOBSET** pair represented in the **formatted IDSET**, the following needs to be added to the serialization buffer in lowest to highest REPLID order.

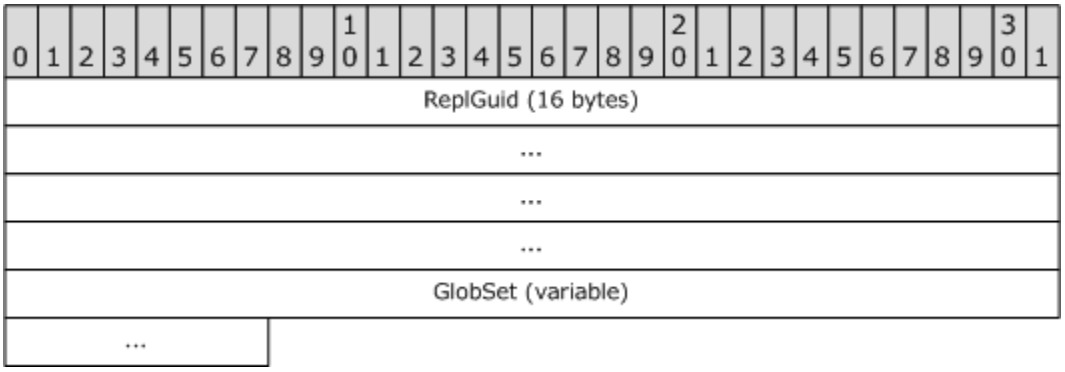


ReplId (2 byte): A REPLID value that when combined with all **GLOBCNT** values represented in the **GlobSet** field, produces a set of IDs.

GlobSet (variable): A serialized GLOBSET.

2.2.2.3.2 Serialized IDSET with REPLGUID

For every **REPLGUID** and **GLOBSET** pair represented in the **formatted IDSET**, the following needs to be added to the serialization buffer.



ReplGuid (16 bytes): A **GUID** value that represents a **REPLGUID**. When combined with all **GLOBCNT** values represented in the **GlobSet** field, produces a set of **GIDs**. The GUID values can be converted into a **REPLID** to produce a set of IDs.

GlobSet (variable): A serialized GLOBSET.

2.2.2.4 GLOBSET

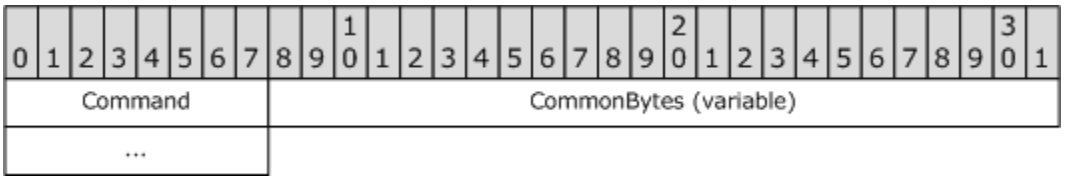
A **GLOBSET** is a set of **GLOBCNT** values that are typically reduced to GLOBCNT ranges.

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 **command**s. For details about how to translate an abstract data model for a GLOBSET into a set of commands, see section 3.1.1.3.

2.2.2.4.1 Push Command (0x01 – 0x06)

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

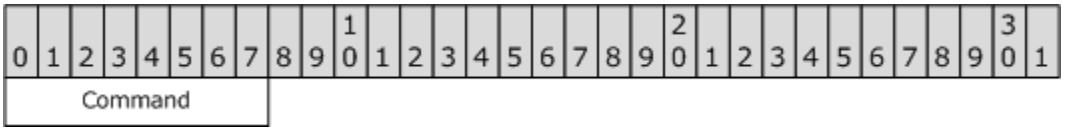


Command (byte): A value in the range "0x01" through "0x06".

CommonBytes (variable): Variable length byte array to be pushed onto the common byte stack. The length of the byte array is equal to the **Command** value ("0x01" through "0x06").

2.2.2.4.2 Pop Command (0x50)

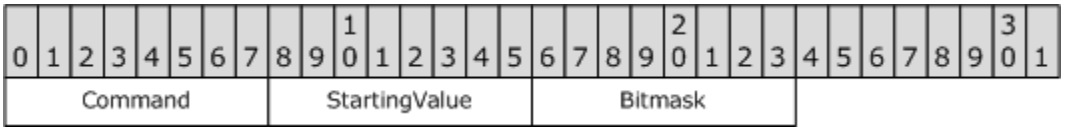
The **Pop** command will remove bytes that were added to the **common byte stack** from the previous **Push** command.



Command (1 byte): The value 0x50.

2.2.2.4.3 Bitmask Command (0x42)

The **Bitmask** command allows for up to five **GLOBCNT** ranges to be compressed into a single encoding command if they all have five high-order bytes in common and the low-order bytes are all within eight values of each other.



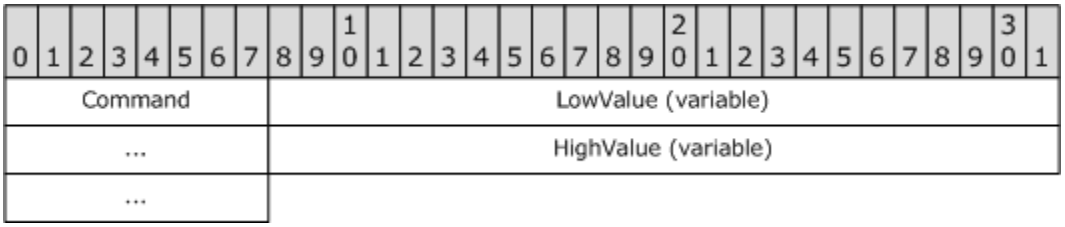
Command (1 byte): The value "0x42".

StartingValue (1 byte): Low-order byte of first GLOBCNT.

Bitmask (1 byte): Creates additional GLOBCNT values that are defined based on which bits are set in **Bitmask**.

2.2.2.4.4 Range Command (0x52)

The **Range** command is used to add a **GLOBCNT** range to the **GLOBSET**. The range is determined by the GLOBSET value produced from the **LowValue** field and the GLOBCNT produced from the **HighValue** field.



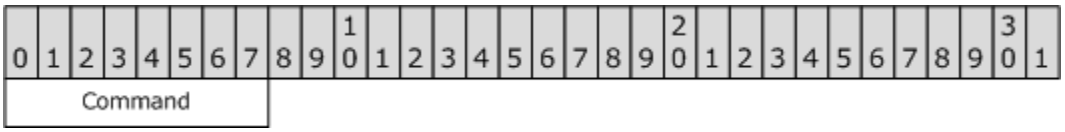
Command (1 byte): The value "0x52".

LowValue (variable): Variable length byte array of low-order values for GLOBCNT generation. The number of bytes in this field is equal to six minus the number of high-order bytes in the **common byte stack**. MUST be less than or equal to **HighValue**, when compared byte to byte.

HighValue (variable): Variable length byte array of low-order values for GLOBCNT generation. The number of bytes in this field is equal to six minus the number of high-order bytes in the common byte stack. MUST be greater or equal to **LowValue**, when compared byte to byte.

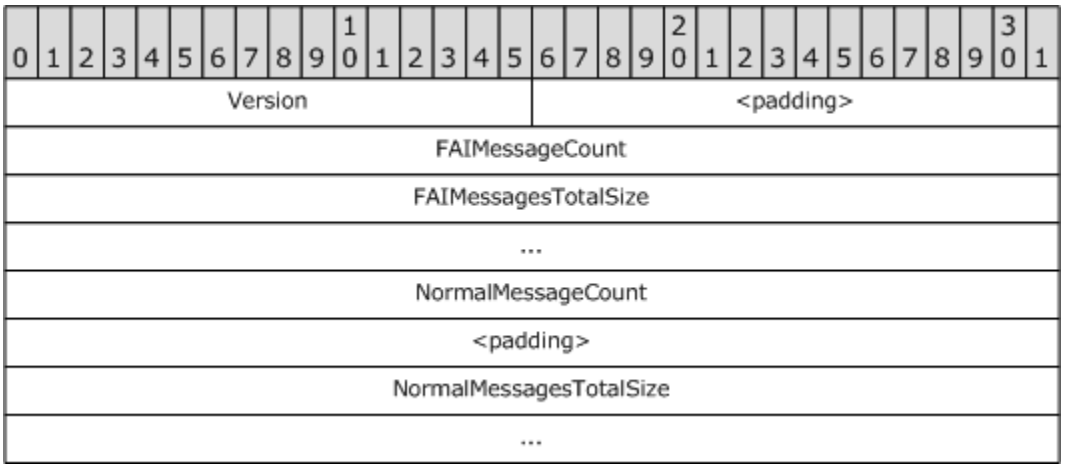
2.2.2.4.5 End Command (0x00)

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



Command (1 byte): The value "0x00".

2.2.2.5 ProgressInformation



Version (2 bytes): A four-bit value that contains a number that identifies the binary structure of the data that follows. The table in this section describes a format for version "0x0000", which is the only version of this structure defined for this protocol.

<padding>: SHOULD be set to zeroes 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.

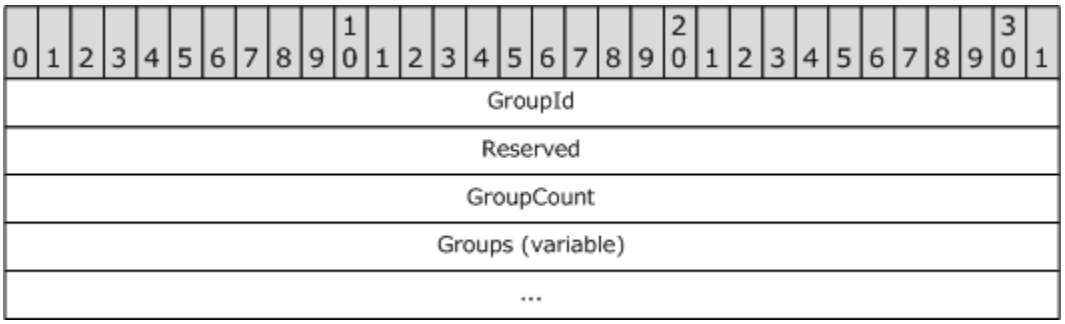
FAIMessageTotalSize (8 bytes): An unsigned 64-bit integer value that contains the size in bytes of all changes to FAI **messages** that are scheduled for download during the current synchronization operation.

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.

NormalMessageTotalSize (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.

2.2.2.6 PropertyGroupInfo

The **PropertyGroupInfo** structure describes a single **property** mapping –between group indexes and **property tags** within a property group. For more details about property groups, see section3.1.1.2.



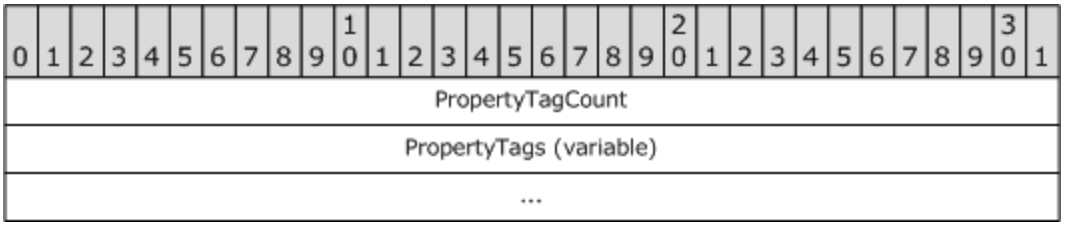
GroupId (4 bytes): An unsigned 32-bit integer value that identifies a property mapping within the current synchronization **download**.

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 are present in the **Groups** field.

Groups (variable): An array of **PropertyGroup** structures. This field MUST contain **GroupCount PropertyGroup** elements.

2.2.2.6.1 PropertyGroup

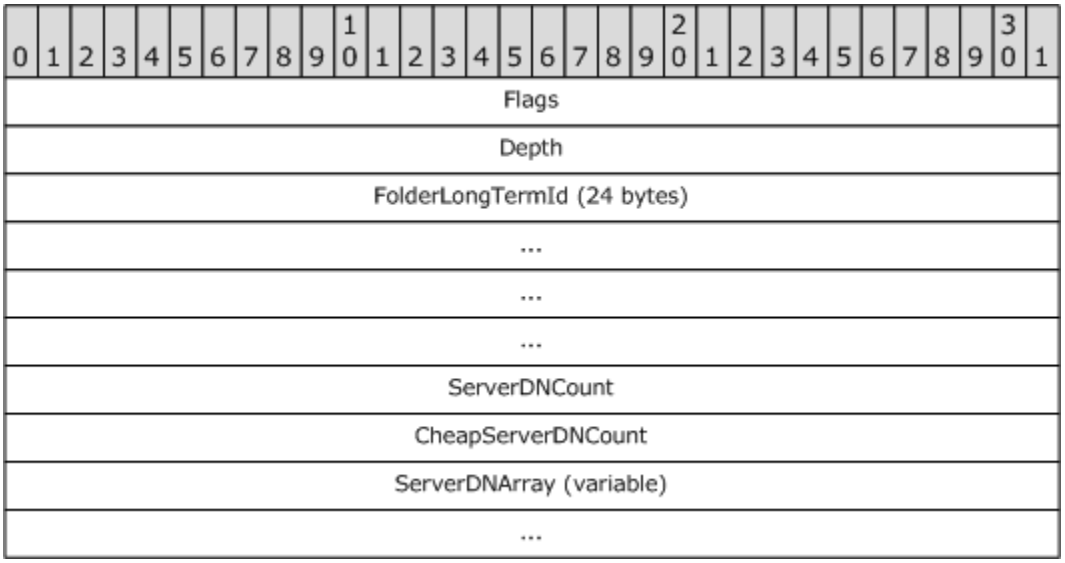


PropertyTagCount (4 bytes): An unsigned 32-bit integer value that specifies how many **property tags** are present in **PropertyTags**.

PropertyTags (variable): An arrray of **PropertyTag** structures. This field MUST contain **PropertyTagCount** tags.

2.2.2.7 FolderReplicaInfo

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



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

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

FolderLongTermId (24 bytes): A **LongTermId** structure. Contains the **LongTermId** of a **folder**, for which replica information is being described.

ServerDNCount (4 bytes): An unsigned 32-bit integer value that determines how many elements exist in **ServerDNArray**.

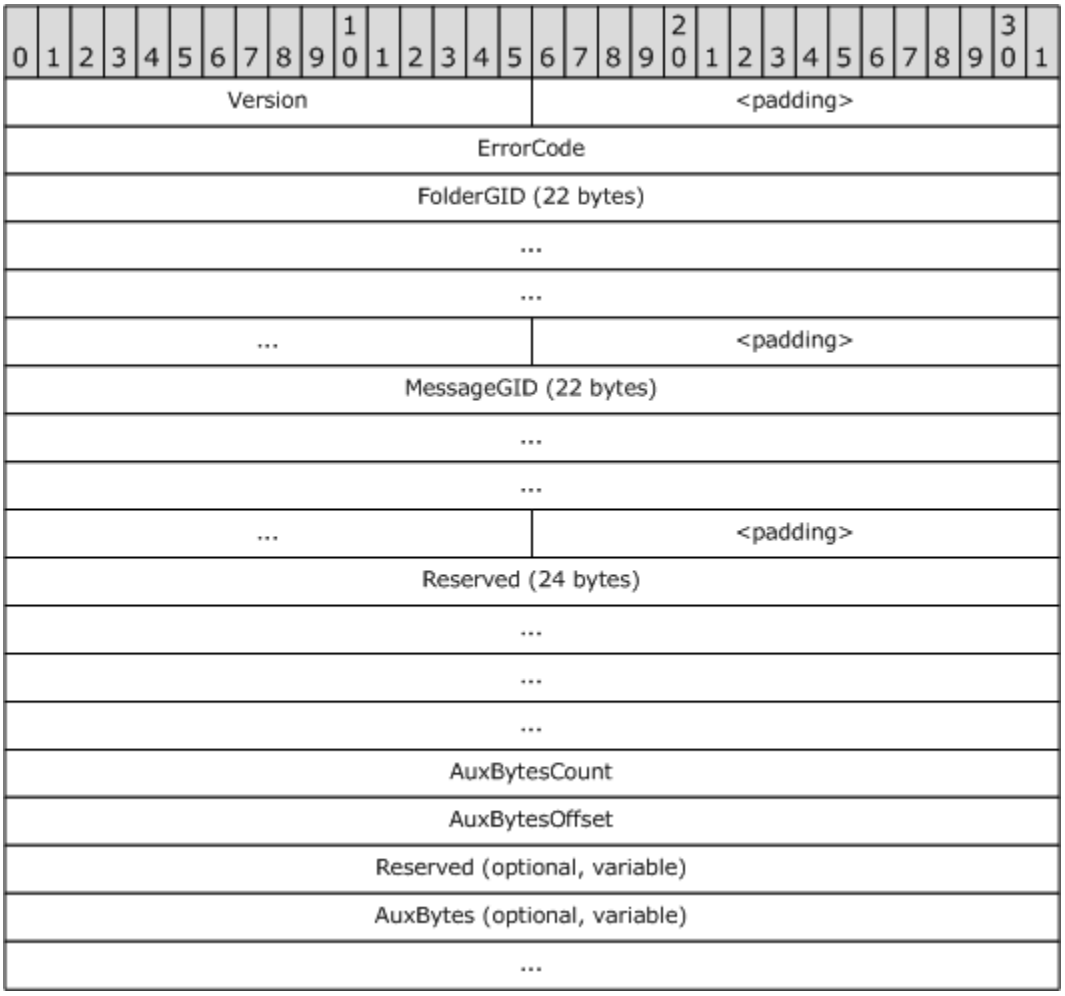
CheapServerDNCount (4 bytes): An unsigned 32-bit integer value that determines how many of the leading elements in **ReplicaMdbArray** have the same, lowest, network access cost. **CheapServerDNCount** MUST be less than or equal to **ServerDNCount**.

ServerDNArray (variable): An array of ASCII-encoded NULL-terminated strings. MUST contain **ServerDNCount** strings. Contains an **enterprise/site/server distinguished name (ESSDN)** of servers that have a replica of the folder identifier by **FolderLongTermId**.

2.2.2.8 ExtendedErrorInfo

Contains extended and contextual information about an error that has occurred when producing a **FastTransfer stream**.

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



Version (2 bytes): An unsigned 16-bit integer that determines the format the structure. The format shown above corresponds to version "0x00000000", which is the only version defined for the protocol. Servers MUST output this structure in a version that corresponds to a version of a protocol chosen by the client.

<padding>: SHOULD be set to zeroes and MUST be ignored by the clients.

ErrorCode: One of the error codes defined in [MS-OXCDATA] that describes the reason for the failure.

FolderGID (22 bytes): A **GID** structure that identifies the **folder** that was in context at the time the error occurred. MUST be filled with zeroes, if no folders were in context.

MessageGID (22 bytes): A GID structure that identifies the **message** that was in context at the time the error occurred. MUST be filled with zeroes, if no messages were in context.

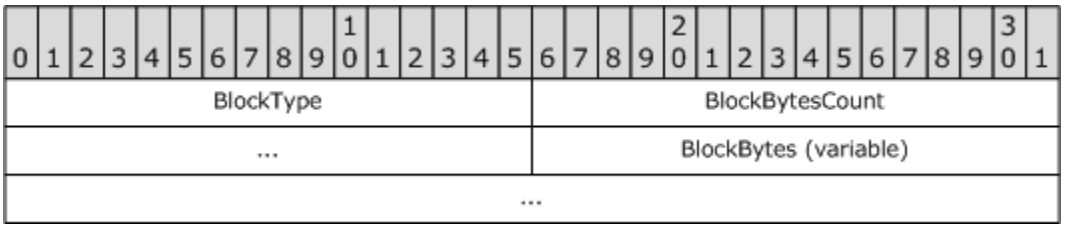
AuxBytesCount (4 bytes): An unsigned 32-bit integer value that specifies the size of the **AuxBytes** field. If set to 0, **AuxBytes** is missing.

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

Reserved (variable): SHOULD be set to zeroes and SHOULD be ignored by clients.

AuxBytes (optional, variable): A **PtypBinary** value that MUST be present and reside at offset **AuxBytes** from the beginning of the structure, **IFF AuxBytesCount** > 0. If present, MUST consist of one or more **AuxBlock** structures serialized sequentially without any padding.

2.2.2.8.1 AuxBlock



BlockType (2 bytes): An unsigned 16-bit integer that specifies the format of the **BlockBytes** field. The known types are described in the following table:

0x0000	Exchange Server diagnostic context (opaque)

BlockBytesCount (4 bytes): An unsigned 32-bit integer value that specifies the size in bytes of the **BlockBytes** field.

BlockBytes (variable): A **PtypBinary** value. Semantics are determined by the value of the **BlockType** field. MUST be exactly **BlockBytesCount** bytes long.

Clients MUST ignore any **AuxBlock** structures whose **BlockType** they do not recognize. Unknown **AuxBlocks** can be easily skipped over to subsequent blocks, because their size can always be determined based on **BlockBytesCount**.

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 MAY also be used ICS operations. See ROP details provided in this section and the following table for more details.

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. Transmission of **messaging object** data based on the context configuration.
3. Checkpointing. An optional step in which data that is required for subsequent initialization of the next iteration of this operation is **downloaded**.
4. Release of resources. Release of resources held on a server. This includes releasing the context by using **RopRelease**.

Note that the context in step 1 is not a messaging object, which means that it is not persisted in a **mailbox** and 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 this section for usage directions.

Operation	Initialization	Data transmission	Checkpointing
FastTransfer download	**RopFastTransfer** **- SourceCopy*** **RopTellVersion**	**RopFastTransfer** **- SourceGetBuffer** Mailbox data is encoded into a **FastTransfer stream**.	Not applicable.
FastTransfer **upload**	**RopFastTransfer** **- DestinationConfigure** **RopTellVersion**	**RopFastTransfer** **- DestinationPutBuffer** Mailbox data is encoded into a FastTransfer stream.	Not applicable.
ICS download	**RopSynchronization** **- Configure** **- UploadStateStream***	**RopFastTransfer** **- SourceGetBuffer** Mailbox data is encoded into a FastTransfer stream.	**RopSynchronization** **- GetState** **RopFastTransfer** **- SourceGetBuffer**

Operation	Initialization	Data transmission	Checkpointing
			The **final ICS state** is downloaded as a part of data transmission
ICS upload	**RopSynchronization** **- OpenCollector** **- UploadStateStream***	**RopSynchronization** **- Import*** ROPs that operate on a **Message object**.	**RopSynchronization** **- GetState** **RopFastTransfer** **- SourceGetBuffer**

In this section, whenever the applicability of a ROP or protocol details are discussed, operations to which an explanation applies will usually be 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

The **FastTransfer stream** is specified in section 2.2.4.

2.2.3.1 Fast Transfer Copy Operations

2.2.3.1.1 Download

The following steps MUST be taken by a client to **download** copies of **messaging objects** from the server in FastTransfer mode.

1. Obtain a **handle** to a messaging object whose contents are requested, or a handle to a messaging object that the client will download a copy of.

2. Send the **RopFastTransferSourceCopy*** request to create a FastTransfer download context on the server and define the parameters and the scope of the operation.
3. Optionally, send a **RopTellVersion** request, if performing a server-to-client-to-server **upload** (as specified in section 3.3.4.1.2.1).
4. Iteratively send **RopFastTransferSourceGetBuffer** requests on the FastTransfer context to retrieve the **FastTransfer stream** with serialized messaging objects.
5. Send a **RopRelease** request to release the messaging object and FastTransfer context obtained in steps 1 and 2.

2.2.3.1.1.1 RopFastTransferSourceCopyTo

RopFastTransferSourceCopyTo initializes a FastTransfer operation to **download** content from a given **messaging object** and its descendant **subobjects**.

The object output in **OutputServerObject** field MUST be released using **RopRelease** as soon as the client no longer needs it.

Request:

InputServerObject: MUST be either an **Attachment**, or a **Message**, or a **Folder object**.

Level (1 byte): An unsigned 8-bit integer. Set to "0" if all descendant subobjects have to be included in the copying, unless explicitly excluded in **PropertyTags**. Set to non-zero if all descendant subobjects have to be excluded from copying. Note that this field MUST NOT be considered when determining what properties and subobjects to copy for descendant subobjects of **InputServerObject**.

CopyFlags (4 byte): A 32-bit flags structure. For more details about the possible values of this structure, see section 2.2.3.1.1.1.1.

SendOptions (1 byte): An 8-bit flag structure. For more details about possible values for this structure, see section 2.2.3.1.1.1.2.

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

PropertyTags (variable): An array of **PropertyTag** structures. Specifies properties and subobjects (as specified in section 2.2.1.6) to exclude when copying a messaging object pointed to by the **InputServerObject**. Note that this field MUST NOT be considered when determining what properties and subobjects to copy for descendant subobjects of **InputServerObject**. See section 3.2.4.6 for more details about the effect of **property** and subobject filters on download operations.

Response:

ReturnValue: An unsigned 32-bit integer. This value represents the **ROP** execution status.

OutputServerObject: MUST be the FastTransfer download context. MUST be present **IFF ReturnValue** equals **Success**.

Remarks:

If **InputServerObject** is a **folder** that was opened to show soft-deleted **messages**, the scope of an operation that this ROP initiates will only include soft-deleted messages. Otherwise, only normal, non-deleted messages will be included. This applies at all levels that are permitted by the **Level** field.

The **Level** field MUST be ignored and treated as if it is set to "0" if **InputServerObject** is an **Attachment object**.

2.2.3.1.1.1.1 CopyFlags

Defines parameters of the FastTransfer **download** operation.

Servers MAY fail the command if unknown flag bits are set.

Name	Value	Description
Move	0x00000001	MUST NOT be passed if **InputServerObject** is not a **folder** or a **message**. If this flag is set, the client identifies the FastTransfer operation being configured as a logical part of a larger object move operation. If this flag 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. See section 3.2.4.4.1 for more server details.
Unused1	0x00000002	MUST be ignored by the server.
Unused2	0x00000004	MUST be ignored by the server.
Unused3	0x00000008	MUST be ignored by the server.
Unused4	0x00000200	MUST be ignored by the server.
Unused5	0x00000400	MUST be ignored by the server.
BestBody	0x00002000	MUST NOT be passed if **InputServerObject** is not a **message**. If set, the server SHOULD output the message body, and the body of embedded messages, in its original format.

		If not set, the server MUST output message body in the compressed **Rich Text Format (RTF)**.

2.2.3.1.1.1.2 SendOption

Defines the parameters of a **download** operation that relate to data representation.

Name	Value	Description
Unicode	0x01	See the following table for all possible combinations of encoding flags. When used on **RopSynchronizationConfigure**, MUST match the value of the **Unicode SynchronizationFlag** (as specified in section 2.2.3.2.1.1.2).
ForUpload	0x03	Used in FastTransfer operations only when the client requests a **FastTransfer stream** with the intent of **uploading** it immediately to another destination server. The **ROP** that uses this flag MUST be followed by **RopTellVersion**. See section 3.3.4.1.2.1 for details about how this affects behaviors of servers and clients.
RecoverMode	0x04	Used when a client supports recovery mode and requests that a server MUST attempt to recover from failures to download changes for individual **messages**. MUST NOT be set when **ForUpload** flag is set.
ForceUnicode	0x08	See the following table for all possible combinations of encoding flags.
Partial	0x10	MUST NOT be passed for anything but **contents synchronization** download. This flag is set if a client supports partial message downloads. If a server supports this mode, it SHOULD output partial message changes if it reduces the size of the produced stream.<4>

Servers MUST <5> fail the ROP if any unknown flag bits are set.

The following table lists all valid combinations of the **Unicode** | **ForceUnicode** flags.

Flag	Description
0	String properties MUST be output in the codepage set in **RopLogon**.
Unicode	String properties MUST be output either in **Unicode**, or in the codepage set on the current logon, with Unicode being preferred.
Unicode | ForceUnicode	String properties MUST be output in Unicode

2.2.3.1.1.2 RopFastTransferSourceCopyProperties

RopFastTransferSourceCopyProperties initializes a FastTransfer operation to **download** content from a given **messaging object** and its descendant **subobjects**.

The object output in the **OutputServerObject** field MUST be released using **RopRelease** as soon as the client no longer needs it.

Request:

InputServerObject: MUST be either an **Attachment**, **Message**, or **Folder object**.

Level (1 byte): An unsigned 8-bit integer. Set to 0 if descendant subobjects have to be included in the copying, if explicitly included in **PropertyTags**. Set to non-zero if all descendant subobjects have to be excluded from copying. Note that this field MUST NOT be considered when determining what properties and subobjects to copy for descendant subobjects of **InputServerObject**.

CopyFlags (1 byte): An 8-bit flag structure. The possible values for this structure are defined in section 2.2.3.1.1.2.1.

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.

PropertyTags (variable): An array of **PropertyTag** structures. This array specifies the properties and subobjects (as specified in section 2.2.1.6) to copy from the **messaging object** pointed to by the **InputServerObject**. Note that this field MUST NOT be considered when determining what properties and subobjects to copy for descendant subobjects of **InputServerObject**.

Response:

ReturnValue: An unsigned 32-bit integer. This value represents the **ROP** execution status.

OutputServerObject: MUST be the FastTransfer download context. MUST be present **IFF ReturnValue** equals **Success**.

Remarks:

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

- **PropertyTags** specify a list of properties and subobjects to include, as opposed to exclude.
- **BestBody** logic SHOULD NOT be used when copying **messages**.

2.2.3.1.1.2.1 CopyFlags

Defines parameters of the FastTransfer **download** operation.

Servers MAY fail the command if unknown flag bits are set.

Name	Value	Description
Move	0x01	MUST NOT be passed if **InputServerObject** is not a **folder** or a **message**. If this flag is set, the client identifies the FastTransfer operation being configured as a logical part of a larger object move operation. If this flag 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. See section 3.2.4.4.1 for more server details.
Unused1	0x02	MUST be ignored by the server.
Unused2	0x04	MUST be ignored by the server.
Unused3	0x08	MUST be ignored by the server.

2.2.3.1.1.3 RopFastTransferSourceCopyMessages

RopFastTransferSourceCopyMessages initializes a FastTransfer operation for **downloading** content and descendant **subobjects** for **messages** identified by a given set of IDs.

The object output in **OutputServerObject** field MUST be released using **RopRelease** as soon as the client no longer needs it.

Request:

InputServerObject: MUST be a **Folder object**.

MessageIdCount (2 bytes): An unsigned 16-bit integer. This value specifies the number of identifiers in the **MessageIds** field. MUST be greater than 0.

MessageIds (variable): An array of 64-bit identifiers. This list specifies the **MIDs** of the messages to copy. **Message**s MUST be contained by a **folder** identified by **InputServerObject**.

CopyFlags (1 byte): An 8-bit flag structure. The possible values for this structure are defined in section 2.2.3.1.1.3.1.

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.

Response:

ReturnValue: An unsigned 32-bit integer. This value represents the **ROP** execution status.

OutputServerObject: MUST be the FastTransfer download context. MUST be present IFF **ReturnValue** equals **Success**.

2.2.3.1.1.3.1 CopyFlags

Defines parameters of the FastTransfer **download** operation.

Servers MAY fail the command if unknown flag bits are set.

Name	Value	Description
Move	0x01	MUST NOT be passed if **InputServerObject** is not a **folder**. If this flag is set, the client identifies the FastTransfer operation being configured as a logical part of a larger object move operation. If this flag 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. See section 3.2.4.4.1 for more server details.
Unused1	0x02	MUST be ignored by the server.
Unused2	0x04	MUST be ignored by the server.
Unused3	0x08	MUST be ignored by the server.

BestBody	0x10	If set, the server SHOULD output the message body, and the body of embedded messages, in their original format. If not set, the server MUST output message bodies in the compressed **Rich Text Format** (**RTF**).

2.2.3.1.1.4 RopFastTransferSourceCopyFolder

RopFastTransferSourceCopyFolder initializes a FastTransfer operation to **download** properties and descendant **subobjects** for a specified **folder**.

The object output in **OutputServerObject** field MUST be released using **RopRelease** as soon as the client no longer needs it.

Request:

InputServerObject: MUST be a **Folder object**.

CopyFlags (1 byte): An 8-bit flag structure. The possible values for this structure are defined in section 2.2.3.1.1.4.1.

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.

Response:

ReturnValue: An unsigned 32-bit integer. This value represents the **ROP** execution status.

OutputServerObject: MUST be the FastTransfer download context. MUST be present IFF **ReturnValue** equals **Success**.

Remarks:

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

- The type of the **InputServerObject** 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 **CopyFlag** field indicates whether to copy subfolders.
- **BestBody** logic SHOULD NOT be used when copying messages.

2.2.3.1.1.4.1 CopyFlags

Defines parameters of the FastTransfer **download** operation.

Servers MAY fail the command if unknown flag bits are set.

Name	Value	Description
Move	0x01	If this flag is set, the client identifies the FastTransfer operation being configured as a logical part of a larger object move operation. If this flag 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. See section 3.2.4.4.1 for more server details.
Unused1	0x02	MUST be ignored by the server.
Unused2	0x04	MUST be ignored by the server.
Unused3	0x08	MUST be ignored by the server.
CopySubfolders	0x10	This flag identifies whether subfolders of a folder specified in **InputServerObject**, MUST be recursively included into the scope.
NoGhostedContent	0x20	If this flag is set, the server SHOULD NOT send **ghosted folder** content.

2.2.3.1.1.5 RopFastTransferSourceGetBuffer

RopFastTransferSourceGetBuffer downloads the next portion of a **FastTransfer stream** that is produced by a previously configured **download** operation.

Request:

InputServerObject: MUST be a download context.

BufferSize (2 bytes): An unsigned 16-bit integer. This field specifies the maximum amount of data (in bytes) to be output in the **TransferBuffer**. If this value is "0xBABE", the server determines the buffer size based on the residual size of the **RPC** buffer.

Clients SHOULD set this to a sentinel value of "0xBABE" to achieve maximum efficiency.

MaximumBufferSize (2 bytes, optional): An unsigned 16-bit integer that specifies the maximum size limit when the server determines the buffer size.

MUST be present IFF **BufferSize** is set to a sentinel value of "0xBABE".

Clients SHOULD set this value to at least the size of the output RPC buffer to achieve maximum efficiency.

Response:

ReturnValue: An unsigned 32-bit integer. This value represents the **ROP** execution status. The following table lists error codes that clients SHOULD implement special handling for.

Name	Description
ServerBusy	The client MUST wait at least the period of time specified in **BackoffTime** before retrying the ROP. Servers MUST NOT output this error code if the client did not indicate <6> that is supports **BackOff** on connect. For more details about version checking, see [MS-OXCRPC] section 3.1.9.

TransferStatus (2 bytes): A 16-bit enumeration. The possible values for this enumeration are defined in section 2.2.3.1.1.5.1.

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

Reserved (1 byte): 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. MUST be present **IFF** the error code is not **ServerBusy**.

BackoffTime (4 bytes, optional): An unsigned 32-bit integer that contains the time, in milleseconds, that a client MUST wait before retrying the ROP. MUST be present IFF the error code is **ServerBusy**.

Remarks:

To obtain all data output by an operation, this ROP 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 Done** or **Error**.

RopFastTransferSourceGetBuffer supports packed buffers, as specified in [MS-OXCRPC] section 3.1.7.2.

If **BufferSize** is set to a sentinel value of "0xBABE", the server MUST limit the amount of data returned in **TransferBuffer** to the residual size of the output buffer minus result structure overhead, or **MaxBufferSize**, whichever is smaller.

The value of **BufferSize**, if it is set to a value other than sentinel value of "0xBABE", has the following semantics:

- The server MUST fail the command before processing by doing the following:
  - Failing the entire RPC with **ecBufferTooSmall** if it will not be able to fit the resulting **BufferSize** bytes in **TransferBuffer** into the biggest possible output RPC buffer allowed by the protocol.
  - Returning **RopBufferTooSmall** if it will not be able to fit the resulting **BufferSize** bytes in **TransferBuffer** into the residual output RPC buffer.
- The server MUST output at most **BufferSize** bytes in **TransferBuffer** even if more data is available.
- The server returns less than or equal to the **BufferSize** bytes in **TransferBuffer**.

2.2.3.1.1.5.1 TransferStatus

Represents the status of the **download** operation after producing data for the **TransferBuffer** field.

Value	Bit	Description
Error	0x0000	The download stopped because a non-recoverable error has occurred when producing a **FastTransfer stream**. The **ReturnValue** field of the **ROP** output buffer contains a code for that error.
Partial	0x0001	The FastTransfer stream was split, and more data is available. **TransferBuffer** contains incomplete data. See section 2.2.4.1 for restrictions on where FastTransfer streams SHOULD be split.
NoRoom	0x0002	
Done	0x0003	This was the last portion of the FastTransfer stream.

2.2.3.1.1.6 RopTellVersion

RopTellVersion 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 3.3.4.1.2.1).

Request:

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 3.1.9.

Response:

ReturnValue: An unsigned 32-bit integer. This value represents the **ROP** execution status.

2.2.3.1.2 Upload

The following steps MUST be taken by a client to **upload** copies of **messaging objects** to the server in FastTransfer mode:

1. Obtain a **handle** to an object, for which appending or replacing properties and/or **subobjects** is requested.
2. Send **RopFastTransferDestinationConfigure** to create a FastTransfer upload context on the server and define the parameters of the operation.
3. Optionally, send **RopTellVersion** if performing a server-to-client-to-server upload (as specified in section 3.3.4.1.2.1).
4. Iteratively send the **RopFastTransferDestinationPutBuffer** on the FastTransfer context to upload the **FastTransfer stream** with the serialized messaging objects.
5. Send **RopRelease** to release the messaging object and the FastTransfer context obtained in steps 1 and 2.

In step 4, if a client simply re-sends the stream that it is getting through the FastTransfer **download**, it MAY consider using an optimized server-to-client-to-server upload process, as specified in section 3.3.4.1.2.1.

2.2.3.1.2.1 RopFastTransferDestinationConfigure

RopFastTransferDestinationConfigure initializes a FastTransfer operation for **uploading** content encoded in a client-provided **FastTransfer stream** into a **mailbox**.

The object output in the **OutputServerObject** field MUST be released using **RopRelease** as soon as the client no longer needs it.

Request:

InputServerObject: MUST be either an **Attachment**, **Message**, or **Folder object**.

SourceOperation (1 byte): An 8-bit enumeration. The possible values for this enumeration are specified in section 2.2.3.1.2.1.1.

CopyFlags (1 byte): 8-bit flag structure. The possible values for this structure are specified in section 2.2.3.1.2.1.2.

Response:

ReturnValue: An unsigned 32-bit integer. This value represents the **ROP** execution status.

OutputServerObject: MUST be the FastTransfer upload context. MUST be present **IFF ReturnValue** equals **Success**.

Remarks:

Any changes to an object identified by **InputServerObject** are not persisted until **RopSaveChangesMessage** is called.

2.2.3.1.2.1.1 SourceOperation

This enumeration is used to specify the type of data in a **FastTransfer stream** that would be **uploaded** by using **RopFastTransferDestinationPutBuffer** on the FastTransfer upload context that is returned in the **OutputServerObject** field.

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

If a FastTransfer stream to be uploaded is produced by a FastTransfer **download** operation, the client MUST pass a value that corresponds to a **RopFastTransferSourceCopy*** ROP that was used to configure the download operation.

SourceOperation enumeration value	Ordinal value	Corresponding ROP of the FastTransfer download
CopyTo	0x01	**RopFastTransferSourceCopyTo**
CopyProperties	0x02	**RopFastTransferSourceCopyProperties**
CopyMessages	0x03	**RopFastTransferSourceCopyMessages**
CopyFolder	0x04	**RopFastTransferSourceCopyFolder**

Servers MUST stop execution of the **ROP** if an unknown **SourceOperation** value is passed.

2.2.3.1.2.1.2 CopyFlags

Defines parameters of the FastTransfer **upload** operation.

Servers MAY fail the command if unknown flag bits are set.

Name	Value	Description
Move	0x01	MUST NOT be passed if **InputServerObject** is not a **folder** or a **message**.

		If this flag is set, the client identifies the FastTransfer operation being configured as a logical part of a larger object move operation.

2.2.3.1.2.2 RopFastTransferDestinationPutBuffer

RopFastTransferDestinationPutBuffer uploads the next portion of an input **FastTransfer stream** for a previously configured FastTransfer **upload** operation.

Request:

InputServerObject: MUST be a FastTransfer upload context.

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

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.

Response:

ReturnValue: An unsigned 32-bit integer. This value represents the **ROP** execution status.

TransferStatus (2 bytes): A 16-bit enumeration. This value specifies the current status of the transfer.<7>

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

TotalStepCount (2 bytes): An unsigned 16-bit integer that contains the approximate total number of steps <8> to be completed in the current operation. This field is only usable for progress information display.

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

BufferUsedSize (2 bytes): An unsigned 16-bit integer. This value is the buffer size that was used. MAY be less than **TransferDataSize IFF** a ROP failed and **ReturnValue** is not equal to **Success**.

2.2.3.2 Incremental Change Synchronization

The following figure shows the steps involved in **ICS**.

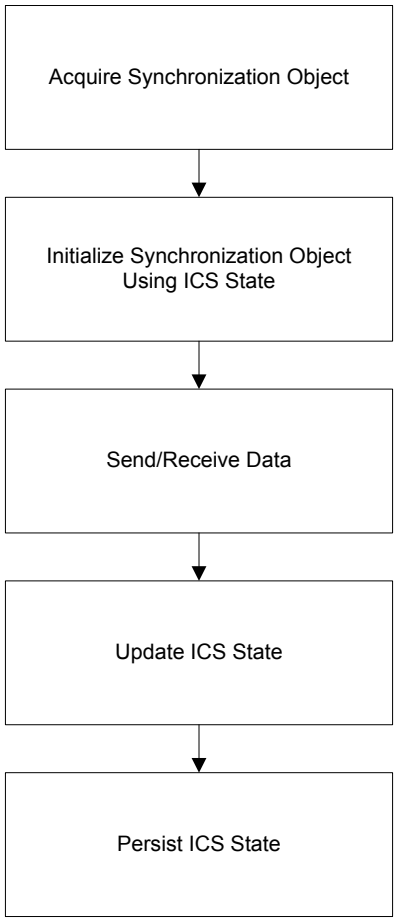


Figure 1: Steps in Incremental Change Synchronization

For a client-centric explanation of how to use this protocol to maintain the **local replica** of a **mailbox**, see [MS-OXCSYNC].

2.2.3.2.1 Download

The following steps MUST be taken by a client when downloading **mailbox** differences from a server:

1. Obtain a **handle** to a **Folder object**, for which synchronization is to be requested. For details about obtaining a **folder handle**, see [MS-OXCFOLD].
2. Send the **RopSynchronizationConfigure** request to create a **synchronization download context** on the server and define the parameters and the scope of the operation.
3. Send the **RopSynchronizationUploadStateStreamBegin/-Continue/-End** requests to **upload** the **initial ICS state** information to the synchronization context.

4. Iteratively send the **RopFastTransferSourceGetBuffer** request on the synchronization context to retrieve the **FastTransfer stream** of the mailbox differences and the **final ICS state**.
5. Persist the ICS state.
6. Send the **RopRelease** request to release the **Folder** object and the synchronization context obtained in steps 1 and 2.

2.2.3.2.1.1 RopSynchronizationConfigure

RopSynchronizationConfigure is used to define the scope and parameters of the synchronization **download** operation. The client MUST **upload** the last remaining piece of configuration data, the **initial ICS state**, before it can request a **FastTransfer stream** that contains differences from the server.

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** and **messages** for **contents synchronization**s).
- A folder that contains these objects as children (contents) or descendants (hierarchy).
- A **restriction** on messages within that folder (contents).

See section 3.3.1.2 for more details.

The object output in **OutputServerObject** field MUST be released using **RopRelease** as soon as the client no longer needs it.

Request:

InputServerObject: 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: content or hierarchy. This field contributes to the synchronization scope. For the possible values for this enumeration, see section 2.2.3.2.1.1.1.

SendOptions (1 byte): An 8-bit enumeration that identifies options for sending the data. For 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. For the possible values of this structure, see section 2.2.3.2.1.1.2.

RestrictionDataSize (2 bytes): An unsigned 16-bit integer that specifies the length of the **RestrictionData** field.

RestrictionData (variable): The variable-length **Restriction** structure, which is used to select the data to be synchronized. This value contributes to the synchronization scope. This field is used in contents synchronization only. The value MUST be set to "0" if **SynchronizationType** is set to Hierarchy ("0x02"). For more details about restrictions, see [MS-OXCDATA].

SynchronizationExtraFlags (4 bytes): A 32-bit flag structure. For the possible values of this structure, see section 2.2.3.2.1.1.3.

PropertyTagCount (2 bytes): An unsigned 16-bit integer that specifies the number of **PropertyTag** structures in **PropertyTags**.

PropertyTags (variable): An array of **PropertyTag** structures (as specified in section 2.2.3.2.1.1.4).

Response:

ReturnValue: 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 **IFF ReturnValue** is **Success**.

2.2.3.2.1.1.1 SynchronizationType

Name	Value	Description
Contents	0x01	Indicates a **contents synchronization**.
Hierarchy	0x02	Indicates a **hierarchy synchronization**.

Servers MUST fail the **ROP** if an unknown **SynchronizationType** value is passed.

2.2.3.2.1.1.2 SynchronizationFlag

Name	Value	Description
Unicode	0x0001	The client supports **Unicode**. The server MUST output values of string properties as they are stored, whether in Unicode or non-Unicode format. This flag MUST match the value of the **Unicode** flag from **SendOptions** field.
NoDeletions	0x0002	The server MUST NOT **download** information about deletions.
NoSoftDeletions	0x0004	MUST NOT be passed for anything but a **contents synchronization** download. The server MUST NOT download information about **messages** that went out of scope. This flag MUST be treated as set if **NoDeletions** is set.

ReadState	0x0008	MUST NOT be passed for anything but a contents synchronization download. The server MUST also download information about changes to the read state of messages.
FAI	0x0010	MUST NOT be passed for anything but a contents synchronization download. The server MUST ignore any changes to **FAI** messages unless this flag is set.
Normal	0x0020	MUST NOT be passed for anything but a contents synchronization download. The server MUST ignore any changes to **normal messages** unless this flag is set.
OnlySpecifiedProperties	0x0080	MUST NOT be passed for anything but a contents synchronization download. If this flag is not set, the server SHOULD exclude properties and **subobjects** output for **folders** and **top-level messages**, if they are listed in **PropertyTags**. If this flag is set, the server SHOULD limit properties and subobjects output for top-level messages to the properties listed in **PropertyTags**.
NoForeignIdentifiers	0x0100	The server MUST ignore any persisted values for the **PidTagSourceKey** and **PidTagParentSourceKey** properties when producing output for folder and message changes. Clients SHOULD set this flag. For more details about possible issues if this flag is not set, see section 3.3.1.1.3.
Reserved	0x1000	MUST be set to "0" when sending. Servers MUST fail the **ROP** request if this flag set.
BestBody	0x2000	MUST NOT be passed for anything but a contents synchronization download. If set, a server SHOULD <9> output message bodies in their original format. If not set, a server MUST output message bodies in the

		compressed **RTF** format.
IgnoreSpecifiedOnFAI	0x4000	MUST NOT be passed for anything but a contents synchronization download. If set, all properties and subobjects of FAI messages MUST be output.
Progress	0x8000	MUST NOT be passed for anything but contents synchronization download. A server SHOULD inject progress information into the output **FastTransfer stream**. This flag is in addition to the means of progress reporting available through the **RopFastTransferSourceGetBuffer** results.

Servers SHOULD <10> fail the ROP if unknown flag bits are set.

2.2.3.2.1.1.3 SynchronizationExtraFlag

Name	Value	Description
Eid	0x00000001	A server MUST include **PidTagFolderId** (for **hierarchy synchronization**) or **PidTagMid** (for **contents synchronization**) into a **folder** change or **message** change header **IFF** this flag is set.
MessageSize	0x00000002	MUST NOT be passed for anything but a contents synchronization **download**. A server MUST include the **PidTagMessageSize property** into a message change header IFF this flag is set.
Cn	0x00000004	A server MUST include the **PidTagChangeNumber** property into a message change header IFF this flag is set.
OrderByDeliveryTime	0x00000008	MUST NOT be passed for anything but a contents synchronization download. The server MUST sort messages by the value of their **PidTagMessageDeliveryTime** property ([MS-OXOMSG] section 2.2.3.9), or by **PidTagLastModificationTime** ([MS-OXCMSG] section 2.2.2.2) if the former is missing, when

		generating a sequence of **messageChange** elements for the **FastTransfer stream**, as specified in section 2.2.4.2.

Servers MUST ignore any unknown flag bits.

2.2.3.2.1.1.4 PropertyTags

Specifies properties and **subobjects** (as specified in section 2.2.1.6) to exclude or include.

This field has different semantics, depending on the value of the **SynchronizationFlag OnlySpecifiedProperties**, as follows:

- If the **OnlySpecifiedProperties** flag is not set, the server SHOULD exclude properties and subobjects from output 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 output for top-level messages to properties listed in the **PropertyTags** field.

In addition to regular **property tags**, this field can contain property tags for the properties that denote **message** subobjects (as specified in section 2.2.4.1.5). Inclusion of these properties in the **PropertyTags** field means that the server SHOULD include or exclude these special parts from output for top-level messages.

2.2.3.2.2 Uploading State

After the synchronization context is acquired, the client MUST supply the **initial ICS State** (as specified in section 2.2.1.1) before executing any other ROPs on it. Depending on the type of the context, the client MUST or SHOULD **upload** the initial ICS state before proceeding. The client MAY choose not to upload the initial ICS state when performing synchronization upload. See section 3.3.4.2.2.1 for details about how that would affect the responsibilities of the roles. The following table summarizes the requirements for the **ICS state** properties being uploaded to different synchronization contexts.

ICS state property	Hierarchy download	Contents download	Hierarchy upload	Contents upload
PidTagIdsetGiven	MUST	MUST	Not applicable.	Not applicable.
PidTagCnsetSeen	MUST	MUST	SHOULD	SHOULD
PidTagCnsetSeenFAI	Not applicable.	MUST	Not applicable.	SHOULD

ICS state property	Hierarchy download	Contents download	Hierarchy upload	Contents upload
PidTagCnsetRead	Not applicable.	MUST	Not applicable.	SHOULD

Uploading the ICS state 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** request, followed by one or more **RopSynchronizationUploadStateStreamContinue** requests. The upload is finished with the **RopSynchronizationUploadStateStreamEnd** ROP.

2.2.3.2.2.1 RopSynchronizationUploadStateStreamBegin

Initiates the **upload** of an **ICS state** property into the synchronization context. No other **property** upload MUST be in progress for this synchronization context, and a property that is being specified in this **ROP** SHOULD NOT have been already uploaded into this synchronization context. This ROP MUST be followed by **RopSynchronizationUploadStateStreamContinue** or **RopSynchronizationUploadStateStreamEnd**.

Request:

InputServerObject: 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 specified in section 2.2.1.1: **PidTagIdsetGiven**, **PidTagCnsetSeen**, **PidTagCnsetSeenFAI**, **PidTagCnsetRead**.

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

Response:

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

2.2.3.2.2.2 RopSynchronizationUploadStateStreamContinue

Continues to **upload** an **ICS state** property value into the synchronization context. This **ROP** MUST be followed by **RopSynchronizationUploadStateStreamContinue** or **RopSynchronizationUploadStateStreamEnd**. Upload MUST be initiated by sending the **RopSynchronizationUploadStateStreamBegin** ROP.

Request:

InputServerObject: MUST be a synchronization context.

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

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

Response:

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

Remarks:

Clients SHOULD skip this ROP if the size of the remaining data specified in **StreamDataSize** field is 0.

2.2.3.2.2.3 RopSynchronizationUploadStateStreamEnd

Concludes the **upload** of an **ICS state** property value into the synchronization context. The upload MUST be initiated by sending a **RopSynchronizationUploadStateStreamBegin** request followed by zero or more iterations of **RopSynchronizationUploadStateStreamContinue**.

Servers concatenate **StreamData** from all received **RopSynchronizationUploadStateStreamContinue** requests for a given ICS state property.

Request:

InputServerObject: MUST be a synchronization context.

Response:

ReturnValue: An unsigned 32-bit integer. This value represents the **ROP** execution status.

2.2.3.2.3 Downloading State

2.2.3.2.3.1 RopSynchronizationGetTransferState

Creates a FastTransfer **download** context for a snapshot of the **checkpoint ICS state** of the operation identified by the given synchronization context.

The object output in **OutputServerObject** MUST be released by using **RopRelease** as soon as the client no longer needs it.

Request:

InputServerObject: MUST be a synchronization context, either download or **upload**.

Response:

ReturnValue: An unsigned 32-bit integer. This value represents the **ROP** execution status.

OutputServerObject: MUST be the **synchronization download context** for the **ICS state**. MUST be present **IFF ReturnValue** equals **Success**.

2.2.3.2.4 Upload

The following steps 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** request to create a synchronization context on the server and to define parameters and the scope of an operation.
3. [Optional] Send the **RopSynchronizationUploadStateStreamBegin/-Continue/-End** request to upload the **initial ICS state** information to the synchronization context.
4. Upload changes, moves, and deletes of individual objects within the mailbox through **RopSynchronizationImport*** ROPs, while passing the synchronization context obtained in step 2.
5. [Optional] Obtain the **final ICS state** by doing the following:
    a. Acquire a separate FastTransfer **download** context for a **checkpoint ICS state** by using **RopSynchronizationGetTransferState** and passing the **synchronization upload context** obtained in step 2 in the request buffer.
    b. Perform the FastTransfer download steps 4-5 (as specified in section 2.2.3.1.1) on the FastTransfer download context acquired in step (a).
    c. Release the FastTransfer download context obtained in step (a).
6. Persist the **ICS state**.
7. Send the **RopRelease** request to release the **Folder object** and the synchronization upload context obtained in steps 1 and 2.

The client MAY elect not to upload/download the ICS states in steps 3 and 5. See section 3.3.4.2.2.1 for details on how that would impact responsibilities of the roles.

When uploading hierarchy differences, the client sends the following **ROP** requests:

- **RopSynchronizationImportHierarchyChange**
- **RopSynchronizationImportDeletes**

When uploading content differences, the client can send any combination of the following ROP requests:

- **RopSynchronizationImportMessageChange.** Imports new **messages** or changes to existing messages.
- **RopSynchronizationImportMessageMove.** Communicates the movement of messages between **folders** within the same mailbox.
- **RopSynchronizationImportDeletes.** Imports deletions of messages.
- **RopSynchronizationImportReadStateChanges.** 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. See [MS-OXCSYNC] section 3.1.5.2 for best practices for ordering different types of upload and download operations.

RopSynchronizationImportMessageChange 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 **ROPSetProperties**, **ROPCreateAttachment**, and so on, followed by **ROPSaveChangesMessage**. For details about additional ROPs, see [MS-OXCROPS] and [MX-OXCMSG].

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	The parent folder never existed.
ObjectDeleted	An object or its parent folder has already been deleted.
IgnoreFailure	The change was ignored, as it has been superseded by another change.

For the complete list of error codes, see [MS-OXCDATA] section 2.4.

2.2.3.2.4.1 RopSynchronizationOpenCollector

RopSynchronizationOpenCollector configures the synchronization **upload** operation, and returns a **handle** to a **synchronization upload context**.

A client SHOULD upload the **initial ICS state** (as specified in section 2.2.3.2.2) into the returned synchronization context prior to using any **RopSynchronizationImport*** ROPs. The client MAY elect not to upload the **initial ICS state**. See section 3.3.4.2.2.1 for details about how that would affect responsibilities of the roles.

The object output in the **OutputServerObject** field MUST be released by using **RopRelease** as soon as the client no longer needs it.

Request:

InputServerObject: 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 3.3.1.2).

IsContentsCollector (1 byte): An 8-bit PtypBoolean value. **TRUE** (non-zero) if synchronization upload is requested for contents of **folders**, and **FALSE** if it is requested for their hierarchy.

Response:

ReturnValue: An unsigned 32-bit integer. This value represents the **ROP** execution status.

OutputServerObject: MUST be the synchronization upload context. MUST be present IFF **ReturnValue** equals **Success**.

2.2.3.2.4.2 RopSynchronizationImportMessageChange

RopSynchronizationImportMessageChange is used to import new **messages** or full changes to existing messages into the **server replica**.

The object output in the **OutputServerObject** field MUST be released using **RopRelease** as soon as the client no longer needs it.

Request:

InputServerObject: MUST be the **synchronization upload context** configured for the collection of changes to content.

ImportFlag (1 byte): An 8-bit flag structure. For details about the possible values for this structure, see section 2.2.3.2.4.2.1.

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

PropertyValues (variable): An array of **PropertyValue** structures. These values are used to specify extra properties on the message, properties that cannot be set using **RopSetProperties**. The following table lists the restrictions that exist for properties passed in this field.

Name	Restrictions	Comments
PidTagSourceKey	Required Fixed position	**GID** of the message being uploaded in the **local replica**.
PidTagLastModificationTime	Required Fixed position	None.
PidTagChangeKey	Required Fixed position	**XID** of a change of a message being uploaded in a local replica. See section 3.1.1.1 for information about how clients can generate this value.
PidTagPredecessorChangeList	Required Fixed position	None.
< *other properties* >	*Prohibited*	None.

Response:

ReturnValue: 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 2.2.3.2.4. The following table contains additional return values.

Name	Description
SyncConflict	A conflict has occurred and **conflict resolution** was either disabled through **ImportFlag FailOnConflict**, or failed. No data was imported.

OutputServerObject: MUST be the **Message object** into which the client will upload the rest of the message changes. MUST be present **IFF ReturnValue** equals **Success**.

MessageId (8 bytes): A 64-bit identifier that specifies the **MID** of the message that was imported. MUST be set to "0x0000000000000000" if the **PidTagSourceKey** property that was passed in **PropertyValues** was a **GID**. MUST be present IFF **ReturnValue** equals **Success**.

Remarks:

The server is responsible for **conflict detection** and resolution, as specified in section 3.1.4.1.

The server MUST detect conflicts. Conflict resolution is controlled by the value of **PidTagResolveMethod** set on the containing **folder**. If a conflict was detected, a ROP MAY succeed and return a **handle** to a **Message object** in the response buffer. The server becomes responsible for performing **conflict resolution** on **RopSaveChangesMessage**, as specified in section 3.1.4.1.2.

2.2.3.2.4.2.1 ImportFlag

Name	Value	Description
Associated	0x10	The **message** being imported is an **FAI**.

Servers MAY fail the ROP if unknown flag bits are set.

2.2.3.2.4.3 RopSynchronizationImportHierarchyChange

RopSynchronizationImportHierarchyChange is used to import new **folders**, or changes to existing folders, into the **server replica**.

Request:

InputServerObject: 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.

HierarchyValues (variable): An array of **PropertyValue** structures. 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 **HierarchyValue** field.

Name	Restrictions	Comments
PidTagParentSourceKey	Required Fixed position	Can be zero-length to identify a folder for which a synchronization **upload** context was opened.
PidTagSourceKey	Required Fixed position	**GID** of the folder being uploaded in the **local replica**.
PidTagLastModificationTime	Required Fixed position	None.
PidTagChangeKey	Required Fixed position	**XID** of a change being uploaded in a local replica. See section 3.1.1.1 for information about how clients can generate its value.
PidTagPredecessorChangeList	Required Fixed position	None.
PidTagDisplayName	Required Fixed position	Value MUST be a non-empty string.
< *other properties* >	*Prohibited*	None.

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

PropertyValues (variable): An array of **PropertyValue** structures. These values are used to specify folder properties.

Response:

ReturnValue: An unsigned 32-bit integer. This value represents the **ROP** execution status. For common return values of **RopSynchronizationImport*** ROPs that require special processing, see section 2.2.3.2.4.

FolderId (8 bytes): A 64-bit identifier. The **FID** of the folder that was imported. MUST be set to "0x0000000000000000" if the **PidTagSourceKey** passed in **PropertyValues** was a **GID**. MUST be present IFF **ReturnValue** equals **Success**.

Remarks:

Changes to parent folders MUST be made before changes to child folders. For example, you cannot send **RopSynchronizationImportHierarchyChange** 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** value of a destination parent folder in the **PidTagParentSourceKey** value in the **HierarchyValues** field while passing the **PidTagSourceKey** value of the folder being moved in the **PidTagSourceKey** property. Moving folders within a public mailbox is not supported.

The server is responsible for **conflict detection** and resolution, as specified in section 3.1.4.1.

If a conflict is detected, the server MUST resolve it as specified in section 3.1.4.1.2 and return **Success**. A server MAY report a conflict using a conflict notification **message**.

2.2.3.2.4.4 RopSynchronizationImportMessageMove

Imports information about moving a **message** between two existing **folders** within the same **mailbox**.

Request:

InputServerObject: 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.

SourceFolderId (variable): An array of bytes. This value contains a serialized representation of the **GID** that represents a **PidTagSourceKey** 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.

SourceMessageId (variable): An array of bytes. This value contains a serialized representation of the GID that represents a **PidTagSourceKey** 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.

PredecessorChangeList (variable): An array of bytes. This value contains a serialized representation of the **PidTagPredecessorChangeList** value 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.

DestinationMessageId (variable): An array of bytes. This value contains a serialized representation of the GID that represents a **PidTagSourceKey** of the message in the destination folder. See section 3.1.1.1 for details about why **DestinationMessageId** MUST be different from **SourceMessageId**.

ChangeNumberSize (4 bytes): An unsigned 32-bit integer. This value specifies the size of the **ChangeNumber** field.

ChangeNumber (variable): An array of bytes. This value contains a serialized representation of the XID that represents a **PidTagChangeKey** of the message in the destination folder.

Response:

Return value: An unsigned 32-bit integer. This value represents the **ROP** execution status. For the common return values of the **RopSynchronizationImport*** ROPs that require special processing, see section 2.2.3.2.4. The following table contains additional return values.

Name	Description
NewerClientChange	The ROP succeeded, but the **server replica** had an older version of a message than the local replica. **ChangeNumber** and **PredecessorChangeList** were not applied to the destination message.

For the complete list of error codes, see [MS-OXCDATA] section 2.4.

MessageId (8 bytes): A 64-bit identifier. The **MID** of the moved message in a destination folder. MUST be set to "0x0000000000000000". MUST be present **IFF ReturnValue** equals **Success**.

Remarks:

Clients MUST <11> only pass folders from private mailboxes in **InputServerObject**.

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

2.2.3.2.4.5 RopSynchronizationImportDeletes

RopSynchronizationImportDeletes imports deletions of **messages** or **folders** into the **server replica**.

Request:

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

IsHierarchy (1 byte): An 8-bit PtypBoolean value. **TRUE** (non-zero) if folder deletions are being imported; otherwise, **FALSE** for message deletions.

PropertyValues (variable): An array of **PropertyValue** structures. The value of this field is used to specify the folders or messages to delete. The following restrictions exist:

Name	Restrictions	Comments
[MVBinary] 0x00001102	Required Fixed position	An array of serialized **GIDs** that represent the objects to be deleted.
< *other properties* >	*Prohibited*	None.

Response:

ReturnValue: An unsigned 32-bit integer. This value represents the **ROP** execution status. For common return values for **RopSynchronizationImport*** ROPs that require special processing, see section 2.2.3.2.4.

2.2.3.2.4.6 RopSynchronizationImportReadStateChanges

Imports **message** read state changes into the **server replica**.

Request:

InputServerObject: 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.

MessageReadStates (variable): An array of **MessageReadState** structures, one per each message that's 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.
- **MessageId (variable):** An array of bytes. Contains a serialized representation of the **XID** that represents a **PidTagSourceKey** for a message that is changing its read state.
- **MarkAsRead (1 byte):** An 8-bit PtypBoolean. This value specifies whether to mark the message as read (**TRUE**, non-zero) or unread (**FALSE**, zero).

Response:

ReturnValue: An unsigned 32-bit integer. This value represents the **ROP** execution status. For common return values for the **RopSynchronizationImport*** ROPs that require special processing, see section 2.2.3.2.4.

Remarks:

MIDs of **FAI** messages in **MessageReadStates** are ignored. This ROP partially succeeds whenever it encounters a problem finding a single message or changing its read state. In case of a partial success, an error code is returned in **ReturnValue**.

2.2.3.2.4.7 RopGetLocalReplicaIds

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 3.3.1.1.1.

Request:

InputServerObject: MUST be a **Logon** object.

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

Response:

ReturnValue: An unsigned 32-bit integer. This value represents the **ROP** execution status.

ReplGuid (16 bytes): A GUID that specifies the **REPLGUID** shared by all allocated IDs.

GlobalCount (6 bytes): An array of bytes. This array specifies the value of the **GLOBCNT** field for the first allocated ID in the allocated set of [**GlobalCount**, **GlobalCount** + **IdCount** – 1].

Remarks:

The client can reconstruct all allocated **GIDs** by combining the returned **ReplGuid** with any GLOBCNT values 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 3.3.1.1.

2.2.3.2.4.8 RopSetLocalReplicaMidsetDeleted

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.

Request:

InputServerObject: 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.

LongTermIdRangeCount (4 bytes): An unsigned 32-bit integer. This value specifies the number of structures in the **LongTermIdRanges** field.

LongTermIdRanges (variable): An array of **LongTermIdRange** structures. Each **LongTermIdRange** structure defines a range of IDs, which are reported as unused or deleted. Consisists of the following:

- **MinLongTermId (24 bytes):** A **LongTermId** structure that defines the ID by using the minimum value of a **GLOBCNT** part that belongs to a range.
- **MaxLongTermId (24 bytes):** A **LongTermId** structure that defines the ID by using the maximum value of a GLOBCNT part that belongs to a range.

The **ReplGuid** parts of **MinLongTermId** and **MaxLongTermId** MUST be the same.

Response:

ReturnValue: An unsigned 32-bit integer. This value represents the **ROP** execution status.

Remarks:

All the IDs contained in **LongTermIdRanges** structures MUST have been obtained previously by using **RopGetLocalReplicaIds**.

RopSetLocalReplicaMidsetDeleted does not deallocate IDs, it only reports that they cannot be used within a given folder. For guidance on the use of **RopSetLocalReplicaMidsetDeleted**, see [MS-OXCSYNC]. See section 3.2.1.2 in this specification for details about its possible application on the server.

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 can be found in section 2.2.4.1.4. The PidTag prefix is omitted to emphasize their difference from properties.

Sections 2.2.4.1 and 2.2.4.2 contain an **ABNF**-like description of the tokenized FastTransfer stream structure. The description uses the conventions established in [RFC4234], except for the following:

- Names enclosed in curly brackets indicate terminal tokens that are serializations of simple types (as specified in section 2.2.4.1.3). They can be followed by *prose* definitions that add restrictions to disambiguate the lexical analysis.
- For display purposes, indented lines represent a continuation of the lines that precede them.

Despite of their name, FastTransfer streams are not represented as Stream objects, and they can only be manipulated by using **RopFastTransferSourceGetBuffer** for **download** operations and **RopFastTransferDestinationPutBuffer** for **upload** operations.

2.2.4.1 Lexical structure

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 **RopFastTransferSourceGetBuffer** or supplied through **RopFastTransferDestinationPutBuffer**. 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 )


namedPropInfo           = {PtypGuid}PropertySet
                             ((%x00 {PtypInteger32}dispid)
                             / (%x01 {PtypString}name))

namedPropId             = {PtypInteger16}PropertyId
                             <Greater or equal to 0x8000>

taggedPropId            = {PtypInteger16}PropertyId
                             <less than 0x8000>

length                  = {PtypInteger32}

fixedPropType           = {PtypInteger16} <see table below>

varPropType             = {PtypInteger16} <see table below>

mvPropType              = {PtypInteger16} <see table below>
```

A FastTransfer stream MAY be larger than a single buffer. The server MUST split the stream when it cannot fit into a single buffer. A stream MUST be split either between two atoms or at any point inside a **varSizeValue**. 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 )
propType                = fixedPropType / varPropType / mvPropType
```

2.2.4.1.1 fixedPropType, varPropType, mvPropType

Property types supported in **FastTransfer streams** are a subset of those defined in [MS-OXCDATA] section 2.13.1.

Lexeme	Range of types defined as a subset of types listed in section 2.14.2 of [MS-OXCDATA]
fixedPropType	Property type value of any type that has a fixed length, as specified in [MS-OXCDATA] section 2.13.1.
varPropType	Property type value of either **PtypString**, **PtypString8** or **PtypBinary**, **PtypServerId**, or **PtypObject.**
mvPropType	Property type value of any multi-valued property type (starts with PtypMultiple), whose base type is either a valid **fixedPropType** or a valid **varPropType**.

2.2.4.1.2 propValue

Represents the identification and a value of a **property** or a **meta-property**.

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

Type	Difference in serialization
PtypBoolean	2-byte 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

Type	Difference in serialization
	FALSE.
PtypUnicode **PtypString8**	Serialization MUST be performed, as specified in [MS-OXCDATA].<12>

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 this table and everywhere else in the document to emphasize their difference from properties.

Start/standalone 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	EndRecip	0x40040003
NewAttach	0x40000003	EndAttach	0x400E0003
Synchronization download			
IncrSyncChg	0x40120003	None.	
IncrSyncDel	0x40130003	None.	

Start/standalone marker name and its numeric value		Corresponding end marker, if applicable, and its numeric value	
IncrSyncEnd	0x40140003	None.	
IncrSyncRead	0x402F0003	None.	
IncrSyncStateBegin	0x403A0003	IncrSyncStateEnd	0x403B0003
IncrSyncProgressMode	0x4074000B	None.	
IncrSyncProgressPerMsg	0x4075000B	None.	
IncrSyncMsg	0x40150003	None.	
Special			
FXErrorInfo	0x40180003		

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 PidTagFXDelProp

A **PtypInteger32** value that represents a directive to a client to delete specific **subobject**s 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 2.2.1.6.

2.2.4.1.5.2 PidTagEcWarning

A **PtypInteger32** value that conveys 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 **PidTagEcWarning** meta-property:

Name	Description
PartiallyComplete	The client SHOULD NOT assume that properties and **subobjects** of an object represented by an element in context were output completely.

For the complete list of error codes, see [MS-OXCDATA] section 2.4.

2.2.4.1.5.3 PidTagNewFXFolder

A **PtypBinary** value that provides information about alternative replicas for a **public folder** in context. Represents a serialized **FolderReplicaInfo** structure.

2.2.4.1.5.4 PidTagIncrSyncGroupId

A **PtypInteger32** value that specifies an identifier of a **property** group mapping. Directs the client to use the specified property group mapping where applicable, until reset with another instance of the **PidTagIncrSyncGroupId** meta-property.

See section 3.1.1.2 for more details about property groups.

2.2.4.1.5.5 PidTagIncrementalSyncMessagePartial

A **PtypInteger32** value that specifies an index of a **property** group within a property group mapping currently in context. Directs a client to treat all forthcoming property values as a part of the specified group, where applicable, until reset with another instance of the **PidTagIncrementalSyncMessagePartial** meta-property.

See section 3.1.1.2 for more details about property groups.

2.2.4.2 Syntactical Structure

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

- Camel-cased names are **non-terminal** syntactic elements ([RFC4234] section 2.3).
- Bolded Pascal-cased names are **markers**. Markers do not have the **PidTag** prefix.
- Normal Pascal-cased names are meta-properties, and have the **PidTag** prefix.

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 sytactical 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
topFolder             = StartTopFld folderContent EndFolder
folderContent         = propList [PidTagEcWarning]
                            ( PidTagNewFXFolder / folderMessages )
                            [ PidTagFXDelProp *subFolder ]
folderMessages        = *2( PidTagFXDelProp messageList )

message               = ( StartMessage / StartFAIMsg )
                            messageContent
                            EndMessage
messageChildren       = [ PidTagFXDelProp *recipient ]
                            [ PidTagFXDelProp *attachment ]
messageContent        = propList messageChildren
messageList           = 1*( [PidTagEcWarning] message )
recipient             = StartRecip propList EndRecip

attachment            = NewAttach 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             = IncrPropertyGroupInfo propList
messageChange         = messageChangeFull / messageChangePartial
```

```
messageChangeFull      = IncrSyncChg messageChangeHeader
                           IncrSyncMsg propList
                           messageChildren
messageChangeHeader    = propList
messageChangePartial   = [groupInfo] [PidTagIncrSyncGroupId]
                           IncrSyncChgPartial messageChangeHeader
                           *( PidTagIncrementalSyncMessagePartial 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

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

Property filters (as specified in section 3.2.4.6) can affect the **Attachment** object properties in the contained **propList**.

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

Name	Restrictions	Comments
PidTagAttachNumber	Required Fixed position	None.
< *other properties* >	None.	None.

2.2.4.3.2 contentsSync

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

See section 3.2.4.1 for details about how servers MUST determine the set of differences that need to be downloaded to clients.

2.2.4.3.3 deletions

The **deletions** element contains information about IDs of **messaging object**s that had been deleted, expired, or moved out of the **synchronization scope** since the last synchronization, as

specified in the **initial ICS state**. See section 3.2.4.1 for details about how servers MUST determine the set of IDs to be reported by using this element.

Deletions SHOULD NOT be present if **SynchronizationFlag NoDeletions** was set when configuring the synchronization **download** operation.

The following restrictions exist on the contained **propList**:

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

Name	Restrictions	Comments
PidTagIdsetDeleted	None.	None.
PidTagIdsetSoftDeleted	Conditional	MUST NOT be present if **SynchronizationType** equals **Hierarchy**. MUST NOT be present if **SynchronizationFlag NoSoftDeletions** is set.
PidTagIdsetExpired	Conditional	MUST NOT be present if **SynchronizationType** equals **Hierarchy**.
< *other properties* >	*Prohibited*	None.

2.2.4.3.4 errorInfo

The **errorInfo** element provides for 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 can be inserted wherever a lexical structure (specified in section 2.2.4.1) allows a **marker** or a **propValue**.

This element SHOULD be used **IFF SendOptions RecoverMode** is set. Note that by the time a server encounters an error that requires failing a **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 **RecoverMode** in **SendOptions**.

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** and **messageList**.

Upon receiving this element, clients MAY perform additional steps to remove a faulty object from future synchronizations, as specified in [MS-OXCSYNC] section 3.1.5.3.3.

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

Name	Restrictions	Comments
[PtypBinary] 0x00000102	Required Fixed position	Serialized **ExtendedErrorInfo** structure. See section 2.2.2.8 for more details.
< other properties >	*Prohibited*	None.

2.2.4.3.5 folderChange

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

The contained **propList** contains the properties of the **Folder object**, possibly affected by **property** filters (as specified in section 3.2.4.6) 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**.

Name	Restrictions	Comments
PidTagParentSourceKey	Required	None.
PidTagSourceKey	Required	None.
PidTagLastModificationTime	Required	None.
PidTagChangeKey	Required	None.
PidTagPredecessorChangeList	Required	None.
PidTagDisplayName	Required	None.
PidTagFolderId	Conditional	MUST be present IFF **SynchronizationExtraFlag Eid** is set.
PidTagParentFolderId	Conditional	MUST be present if **SynchronizationFlag NoForeignIdentifiers** is set.

Name	Restrictions	Comments
< *other properties* >	None.	None.

2.2.4.3.6 folderContent

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

The **propList** contains the properties of the **Folder object**, which are possibly affected by **property** filters (as specified in section 3.2.4.6).

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

Name	Restrictions	Comments
PidTagFolderId	Conditional Fixed position	MUST be present **IFF** the folder is started with **StartTopFld**.
PidTagDisplayName	Required Fixed position	None.
PidTagComment	Required Fixed position	None.
< *other properties* >	None	None.

See section 3.2.4.6 for more details about the impact of **property** and **subobject** filters that are specified when configuring an operation on the content of this element.

The **PidTagEcWarning** meta-property MUST be output by the server if the client does not have the permissions necessary to open the folder, to read its contents, view its subfolder structure, or any additional permissions, as specified in section 3.2.4.4.1. The warning is necessary to make it possible for a client to tell this case from an empty folder.

The **PidTagNewFXFolder** meta-property MUST be output instead of **message** elements when outputting a **public folder** whose contents have not replicated yet.

Under conditions specified in section 3.2.4.6, **subFolder** elements MUST be preceded by a **PidTagFXDelProp** meta-property for the **PidTagContainerHierarchy** property.

2.2.4.3.7 folderMessages

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

All **FAI** messages MUST be output first, followed by **normal messages**. Under conditions specified in section 3.2.4.6, each of these groups MUST be preceded by a **PidTagFXDelProp** meta-property for the corresponding **subobject**, **PidTagFolderAssociatedContents** or **PidTagContainerContents** respectively.

2.2.4.3.8 groupInfo

The **groupInfo** element provides a definition for the **property** group mapping (as specified in section 3.1.1.2). Property group mappings, after they are defined by using the **groupInfo** element, can be referenced with the **PidTagIncrSyncGroupId** meta-property further in the stream by its group ID.

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

Name	Restrictions	Comments
[PtypBinary] 0x00000102	Required Fixed position	Serialized **PropertyGroupInfo** structure. See 2.2.2.6 for more details.
< *other properties* >	*Prohibited*	None.

2.2.4.3.9 hierarchySync

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

See section 3.2.4.1 for details about how servers MUST determine the set of differences that need to be downloaded to clients.

The parent-child relationship is determined by comparing the **PidTagSourceKey** of a prospective parent **folder** and a **PidTagParentSourceKey** of a prospective child folder. The **folderChange** elements with zero-length **PidTagParentSourceKey** values are children of the root of the synchronization 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 **RopSynchronizationConfigure**) 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.

See section 3.2.4.1 for details about how servers MUST determine the set of differences that need to be downloaded to clients.

2.2.4.3.10 message

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.

2.2.4.3.11 messageChange

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

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

- **SendOptions Partial** flag was not set.
- The **MID** of the **message** to be output is not in **PidTagIdsetGiven** from the **initial ICS state**.
- The message is an **FAI** message.
- The message is a conflicting version contained in a conflict resolve message. See section 3.1.4.1.2.1 for details.

Otherwise, it is up to the server to determine the most efficient way to communicate the message change on a case-by-case basis.

2.2.4.3.12 messageChildren

The **messageChildren** element represents children of the **Message objects**: **recipient** and **Attachment objects**.

See section 3.2.4.6 for more details about the impact of **property** and **subobject** filters that are specified when configuring an operation on the content of this element.

Under the conditions specified in section 3.2.4.6, **recipient** and **attachment** elements MUST be preceded by a **PidTagFXDelProp** meta-property for **PidTagMessageRecipients** and **PidTagMessageAttachments** respectively.

2.2.4.3.13 messageChangeFull

The **messageChageFull** 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 3.2.4.6) can affect the **Message object** properties in the contained **propList**.

2.2.4.3.14 messageChangeHeader

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**.

Name	Restrictions	Comments
PidTagSourceKey	Required Fixed position	None.
PidTagLastModificationTime	Required Fixed position	None.
PidTagChangeKey	Required Fixed position	None.
PidTagPredecessorChangeList	Required Fixed position	None.
PidTagAssociated	Required Fixed position	None.
PidTagMid	Conditional	MUST be present **IFF SynchronizationExtraFlag Eid** is set.
PidTagMessageSize	Conditional	MUST be present IFF S**ynchronizationExtraFlag MessageSize** is set.
PidTagChangeNumber	Conditional	MUST be present IFF **SynchronizationExtraFlag Cn** is set.
< other properties >	*Prohibited*	None.

2.2.4.3.15 messageChangePartial

The **messageChangePartial** element <4> represents the difference in **message** content since the last **download**, as identified by the **initial ICS state**. Changes to a message are output based on the granularity of the **property** group (as specified in section 3.1.1.2). The last encountered **PidTagIncrSyncGroupId** meta-property determines which property group mapping MUST be used.

Clients MUST treat every contained **propList** element as the complete content of a property group denoted by the **PidTagIncrementalSyncMessagePartial** meta-property that preceded it. That is, all properties missing from a **propList**, but defined for this group in the corresponding property group mapping, MUST be deleted.

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

Name	Restrictions	Comments
[PtypInteger32] 0x00000003	Conditional	MUST be present **IFF** a property group is empty, but was still marked as changed since the last download. Value MUST be "0". MUST be ignored by clients.
< *other properties* >	None	None.

2.2.4.3.16 messageContent

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

Property filters (as specified in section 3.2.4.6) can affect the **Message object** properties in the contained **propList**.

Name	Restrictions	Comments
PidTagMid	Required Fixed position	Clients MUST ignore the value of this property for embedded messages.
< *other properties* >	None.	None.

2.2.4.3.17 messageList

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 **PidTagEcWarning** if a client does not have the permissions necessary to access it (as specified in section 3.2.4.4.1). 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

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

MUST be present **IFF** the **progessTotal** element was output within the same ancestor **contentsSync** element.

MUST NOT be present if **SynchronizationFlag Progress** was not set when configuring the synchronization **download** operation.

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

Name	Restrictions	Comments
[PtypInteger32] 0x00000003	Required Fixed position	Size of the message to be follow. Servers MAY supply the same value as the **PidTagMessageSize** in **messageChangeHeader,** or use a different approximation.
[Boolean] 0x0000000B	Required Fixed position	Identifies whether the **Message object** that follows **FAI**.
< other properties >	*Prohibited*	None.

2.2.4.3.19progressTotal

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

Note that this method of reporting progress is provided in addition to what is available in the **RopFastTransferSourceGetBuffer** response. 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 **SynchronizationFlag Progress** was set when configuring the synchronization download operation, and a server supports progress reporting.

This element MUST NOT be present if **SynchronizationFlag Progress** was not set when configuring the synchronization download operation.

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

Name	Restrictions	Comments
[PtypBinary] 0x00000102	Required Fixed position	Serialized **ProgressInformation** structure. See section 2.2.2.5 for more details.
< other properties >	*Prohibited*	None.

2.2.4.3.20propList

The **propList** elements MUST NOT contain **propValue** elements for meta-properties. All instances in which meta-properties can be encountered in a document are mentioned explicitly in the syntax **ABNF**.

Syntactic elements that contain a **propList** 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) instead of an actual value MUST be omitted from the **propList**.

2.2.4.3.21 propValue

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 **RopQueryRows** and **RopGetPropertiesSpecific**. Clients and servers MUST be capable of accepting large amounts of data and MUST fail the operation if the size of data crosses the threshold imposed by an implementation, rather than truncating the data.

2.2.4.3.22 readStateChanges

The **readStateChanges** element contains information about **MIDs** of **Message objects** that had their read state changed since the last synchronization, as specified by the **initial ICS state**. See section 3.2.4.1 for details about how servers MUST determine the set of IDs to be reported using this element.

This element SHOULD NOT be present if **SynchronizationFlag ReadState** was not set when configuring the synchronization **download** operation.

The following restrictions exist on the contained **propList**:

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

Name	Restrictions	Comments
PidTagIdsetRead	None.	None.
PidTagIdsetUnread	None.	None.
< *other properties* >	*Prohibited*	None.

2.2.4.3.23 recipient

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

The **propList** child element contains the properties of the **Recipient** object.

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

Name	Restrictions	Comments
PidTagRowid	Required Fixed position	None.
< *other properties* >	None.	None.

2.2.4.3.24 root

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 sections 2.2.4.4 and 2.2.3.1.2.1.1.

2.2.4.3.25 state

The **state** element contains the **final ICS state** of the synchronization **download** operation. See sections 3.2.4.1 and 3.2.1.1 for details about how servers MUST construct the final ICS state.

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

Name	Restrictions	Comments
PidTagIdsetGiven	None.	None.
PidTagCnsetSeen	None.	None.
PidTagCnsetSeenFAI	Conditional	MUST NOT be present if **SynchronizationType** equals **Hierarchy**.
PidTagCnsetRead	Conditional	MUST NOT be present if **SynchronizationType** equals **Hierarchy**.
< *other properties* >	*Prohibited*	None.

2.2.4.4 Applicability to ROPs

The following table describes how possible root elements in the **FastTransfer stream** correspond to Bulk Data Transfer operations defined in section 1.3. 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 2.2.3.1.1 and 2.2.3.2.1).

ROP that initiate an operation	Root element in the produced FastTransfer stream	ROP request buffer field conditions
RopSynchronization-		
- **Configure**	`contentsSync`	**SynchronizationType** equals **Contents**.
	`hierarchySync`	**SynchronizationType** equals **Hierarchy**.
- **GetState**	`state`	Always.
RopFastTranserSource-		
- **CopyTo** - **CopyProperties**	`folderContent`	**InputServerObject** is a **Folder object**.
	`messageContent`	**InputServerObject** is a **Message object**.
	`attachmentContent`	**InputServerObject** is an **Attachment object**.
- **CopyMessages**	`messageList`	Always.
- **CopyFolder**	`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 **RopFastTransferDestinationConfigure**. See section 2.2.3.1.2.1.1 for details about the applicability of FastTransfer streams to FastTransfer upload operations.

3 Protocol Details

3.1 Common Details

3.1.1 Abstract Data Model

The protocol details in sections 3.1.1.1 through 3.1.1.3.3.2.5 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	Union of two sets. Every element in the resulting set belongs to either A, or B, or both.
⋂	A ⋂ B	Intersection of two sets. Every element in the resulting set belongs to both A and B.
{ }	$\{A_1, \ldots, A_n\}$	A set consisting of elements A_1 through A_n.
⊆ ⊇	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	Instructs to include an element into a set. The Set is assigned to Set ⋃ {element}.
ø	A = ø	Empty set: a set that contains no elements. Set A is asserted to be an empty set, it has no elements.

3.1.1.1 Object and Change Identification

On creation, objects in the **mailbox** are assigned internal identifiers, commonly known as **FIDs** for **folders** and **MIDs** 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 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 usually happens during execution of ROPs, such as **RopSaveChangesMessage** and **RopCopyTo**, or while processing events not controlled by the client (such as **Message object** delivery).

Messaging objects also maintain a **change number**, or **CN**, which identifies a version of an object and adheres to the same rules as internal identifiers for messaging objects. A new change number is assigned to an object whenever an object is created or modified. For messages, in addition to a change number for the entire message, there are additional mechanisms for tracking changes to their elements: read state (as specified in section 3.2.4.3) and properties and **subobjects** arranged into groups (as specified in section 3.1.1.2).

A protocol role that generates internal identifiers for messaging objects and changes MUST ensure that the **GLOBCNT** portions of the internal identifiers that share the same **NamespaceGuid** (as specified in the **XID** structure in section 2.2.2.1) 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, as opposed to a synchronization, its **predecessor change list** (**PCL)** MUST be merged with the XID that represents the new change number.

Although it is not recommended as a general practice, it is possible to change an object without altering its change number, and therefore without flagging it for synchronization. For more details about changing an object without altering its change number, see the ROPs specified in [MS-OXCROPS] that end with "NoReplicate".

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 3.3.1.1.1. Clients MUST generate **foreign identifiers** (as specified in section 3.3.1.1.3) to identify client-side changes to objects that they import through ICS **upload**.

Upon successful import of a new or changed object using ICS upload, the server MUST do the following when receiving **RopSaveChangesMessage**:

- Assign the object a new internal change number (**PidTagChangeNumber**).
  This is necessary because the server MUST be able to represent the imported version in the **PidTagCnsetSeen** or **PidTagCnsetSeenFAI** properties, and these properties cannot operate on foreign identifiers for change numbers that a client passes.
- Assign the object an internal identifier (**PidTagMid** or **PidTagFolderId**) based on the kind of **external identifier** that was passed for the objects identification by the client **IFF** the object is new.
  - If the external identifier is a **GID**, the server MUST convert it to a short-term internal identifier and assign it to an imported object.
- Assign the object the given **PidTagChangeKey** and PCL (**PidTagPredecessorChangeList**) 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**. See section 3.1.4.1 for details about handling conflict.

Foreign identifiers supplied by clients for change identification (**PidTagChangeKey**) are replaced whenever their corresponding internal identifiers change. Examples are provided in the following table.

Sequence of client action	Corresponding server reaction
RopSynchronizationImportMessageChange for a new message: • SourceKey = GID(ID1) • ExternalChangeNumber = XCN1 Client checkpoints the stored **initial ICS state**: IdsetGiven += ID2	• SourceKey = GID(ID1) • Mid = ID1 • ExternalChangeNumber = XCN1 • ChangeNumber = CN2 • Final ICS State: CnsetSeen += CN2
RopSynchronizationImportMessageChange • SourceKey = GID(ID1) • ExternalChangeNumber = XCN3	• ExternalChangeNumber = XCN3 • ChangeNumber = CN4 • Final ICS state: CnsetSeen += CN4
ICS download of contents	• SourceKey = GID(ID1) • Mid = ID1 • ExternalChangeNumber = XCN3 • ChangeNumber = CN4
RopOpenMessage – **RopSetProperties** – **RopSaveChangesMessage**	• ChangeNumber = CN5

ICS Download	• Changes to a message:
	  ○ SourceKey = GID(ID1)
	  ○ Mid = ID1
	  ○ ExternalChangeNumber = GID(CN5)
	  ○ ChangeNumber = CN5
	• Final ICS state: CnsetSeen += CN5

Sequence of client action	Corresponding server reaction

RopSynchronizationImportMessageMove	• Message is hard-deleted in the source folder A.
	• A copy of the message is created in destination folder B with:
	  o Mid = ID2
	  o ChangeNumber = CN6

ICS download of contents for folder A	• Deletions: ID1 • Final ICS state: IdsetGiven -= ID1

ICS download of contents for folder B	• New message:
	  o SourceKey = GID(ID2)
	  o Mid = ID2
	  o ExternalChangeNumber = GID(CN6)
	  o ChangeNumber = CN6
	• Final ICS state:
	  o IdsetGiven -= ID2
	  o CnsetSeen += CN6

RopSynchronizationImportMessageChange • SourceKey = GID(ID2) • ExternalChangeNumber = XCN7	• ExternalChangeNumber = XCN7 • ChangeNumber = CN8

3.1.1.2 Property Groups

If servers choose to support <4> partial message change synchronization, they 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 of the **FastTransfer stream**.

ICS is optimized for reporting partial changes to messages on a **property** group basis. The simplest approach for servers providing that information is to track changes made to groups of properties. A group is considered changed if any of the properties in the group are modified. It is up to the server to define a property group mapping - how properties are distributed into

groups. ICS offers a way to communicate property group mapping information per-message, so every message MAY 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 single **attachment** property would mean that all the properties in the attachment property group are updated during ICS. Likewise, a change to any one body property would mean that all the properties in the body 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 3.2.4.3.

How properties are organized into property groups determines their property group mapping. One message in a **mailbox** might have a different mapping <13> than another message, which means that the properties in group N on one message might be different than the properties in group N in another message.

3.1.1.3 Serialization of IDSET

When an **IDSET** has to be transmitted from a client to a server or from a server to a client, it has to be serialized. This section contains details about how IDSETs MUST be serialized.

3.1.1.3.1 Formatted IDSET

Before serialization, the contents of an **IDSET** have to be arranged in such a way as to allow it to be properly encoded. The ID values MUST be arranged by **REPLID** and all IDs for each REPLID MUST be reduced into a **GLOBSET** of **GLOBCNT** values. Each GLOBSET MUST be arranged from lowest to highest GLOBCNT 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 diagram shows what a properly **formatted IDSET** MUST look like for serialization.

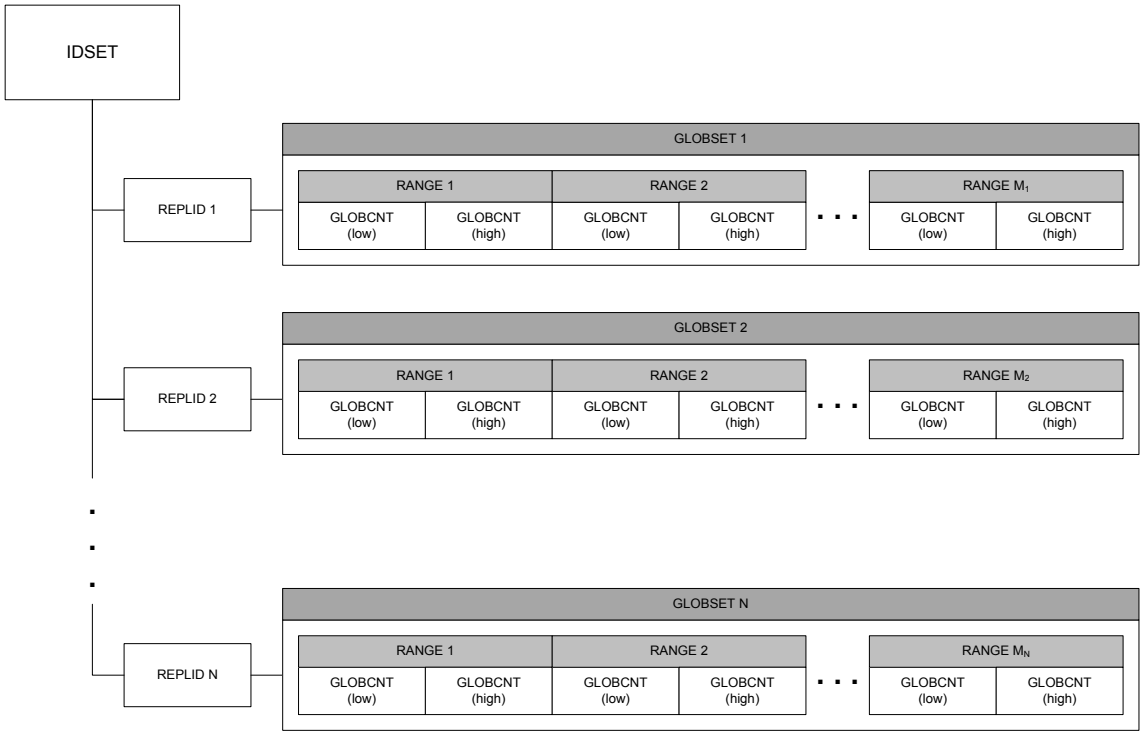


Figure 2: Formatted IDSET

3.1.1.3.2 IDSET Serialization

There are two different formats in which a serialized IDSET can exist on the wire. The only difference is how the **REPLID** value is represented in the serialization buffer. The first format contains the REPLID value followed by the **GLOBSET** data. The second format contains, instead of the REPLID, the **REPLGUID** that is associated with the REPLID, followed by the GLOBSET data. No information contained in the serialized buffer identifies which format is being used. The context in which the serialized IDSET 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 (see 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 (see section 2.2.1.3.1), then it MUST be transmitted in a REPLID-based form. Sections 3.1.1.3.3 through 3.1.1.3.3.2.5 describe the layout of both formats on the wire. REPLID-based format can be converted to REPLGUID-based one by using mapping operations, as described in [MS-OXCSTOR].

See section 2.2.2.3 for more details about the format of each serialized IDSET.

3.1.1.3.3 GLOBSET Serialization

IDSET serialization requires each **GLOBSET** within the IDSET to be serialized. The **GLOBCNT** ranges within the GLOBSET are serialized by using special encoding commands to compress the amount of data for each GLOBCNT pair. This section contains information about how to encode and decode a GLOBSET during IDSET serialization.

3.1.1.3.3.1 Encoding

The following commands can be used to encode a **GLOBSET**.

3.1.1.3.3.1.1 Push Command (0x01 – 0x06)

The **Push** command SHOULD be used when multiple **GLOBCNT** values share the same high-order values. For example, if all GLOBCNT values have the same two high-order bytes, the **Push** command (0x02) SHOULD be used to push two bytes onto the **common byte stack**. These two bytes will be used to create GLOBCNT 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 will place a singleton GLOBCNT range into the **GLOBSET** 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.

See section 2.2.2.4.1 for more details about the format of the **Push** command.

3.1.1.3.3.1.2 Pop Command (0x50)

Bytes that have been pushed onto the **common byte stack** with a **Push** command can be removed using the **Pop** command. 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** values 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.

See section 2.2.2.4.2 for more details about the format of the **Pop** command.

3.1.1.3.3.1.3 Bitmask Command (0x42)

The **Bitmask** command is used when there are multiple **GLOBCNT** ranges 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.

See section 2.2.2.4.3 for more details about the format of the **Bitmask** command and its fields.

The **StartingValue** field MUST be set to the low-order byte of the low value of the first GLOBCNT range. The **Bitmask** field 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 the **StartingValue**. From the result, subtract one. The bit numbers within the **Bitmask** field are 0 for the lowest bit to 7 the highest bit. For all GLOBCNT values between ranges, the bit is not set.

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 as a **StartingValue** of 0x01 and the **Bitmask** would be 0xEB. The **Bitmask** value 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 **StartingValue** and each low-order byte value corresponding to a bit that is set in the **Bitmask**, you end up with the low-order byte values {0x01, 0x02, 0x03, 0x05, 0x07, 0x08, 0x09}. If you collapse these into ranges, you will have {0x01-0x03, 0x05-0x05, 0x07-0x09}.

3.1.1.3.3.1.4 Range Command (0x52)

The **Range** command is used to generate a single **GLOBCNT** range. 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 prior to using the **Range** command. The low-order bytes that are not in common are used to build the **Range** command.

See section 2.2.2.4.4 for more details about the format of the **Range** command and its fields.

3.1.1.3.3.1.5 End Command (0x00)

The **End** command is used to signal the end of the **GLOBSET** encoding. This command MUST be added after all **GLOBCNT** ranges 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 before the **End** command can be used.

See section 2.2.2.4.5 for more details about the format of the **End** command.

3.1.1.3.3.2 Decoding

The following commands can exist in a serialized **GLOBSET**.

3.1.1.3.3.2.1 Push Command (0x01 – 0x06)

The **Push** command can add one to six bytes of high-order bytes to a **common byte stack**. The common byte stack is used in conjunction with subsequent encoding commands to build **GLOBCNT** pairs that represent GLOBCNT ranges within the **GLOBSET**. When building a GLOBCNT, all the bytes on the common byte stack are used and any remaining bytes needed for a complete GLOBCNT have to come from another encoding command. The common bytes are pushed onto the stack highest to lowest byte order.

See section 2.2.2.4.1 for more details about the format of the **Push** command in the serialization buffer.

3.1.1.3.3.2.2 Pop Command (0x50)

The **Pop** command removes the bytes that were previously pushed onto the **common byte stack** from the last **Push** command. The **Pop** command unwinds the stack in the reverse order in which the bytes where pushed.

See section 2.2.2.4.2 for more details about the format of the **Pop** command in the serialization buffer.

3.1.1.3.3.2.3 Bitmask Command (0x42)

The **Bitmask** command MUST only be encountered when there are five bytes in the **common byte stack**.

See section 2.2.2.4.3 for more details about the format of the **Bitmask** command and its fields.

Using the **StartingValue** and the **Bitmask** fields of the **Bitmask** command, a set of low-order bytes can be produced. See section 3.1.1.3.3.1.3 for more details about decoding the **Bitmask** field to produce individual low-order values. Each low-order byte MUST be combined with the required five high-order bytes on the common byte stack or form a complete 6-byte **GLOBCNT** value, which MUST be added to the **GLOBSET**.

3.1.1.3.3.2.4 Range Command (0x52)

The **Range** command generates a **GLOBCNT** range. The GLOBCNT range MUST be added to the **GLOBSET**.

See section 2.2.2.4.4 for details about the format of the **Range** command and its fields.

The **Range** command contains two byte array fields, the **LowValue** and **HighValue**. Each of these fields MUST be combined with any high-order bytes in the **common byte stack** to produce a 6-byte GLOBCNT value. The two GLOBCNT values are the low and high value of the GLOBCNT range.

3.1.1.3.3.2.5 End Command (0x00)

When the **End** command is encountered, the **GLOBSET** MUST be complete based on the **GLOBCNT** values generated from any previous encoding commands. The **End** command MUST NOT be encountered when there are bytes stored on the **common byte stack**.

3.1.2 Timers

None.

3.1.3 Initialization

None.

3.1.4 Higher-Layer Triggered Events

3.1.4.1 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 decide 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 be compatible with the one mandated for servers, as specified in this section, to ensure the consistency of user experience regardless of the role performing the conflict handling. When referring to synchronization in this specification, both **download** and **upload** are considered, unless specified otherwise.

3.1.4.1.1 Detection

Servers MUST implement **conflict detection** using an algorithm compatible with 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 for objects that have the same value for the **PidTagSourceKey** property.

Clients MAY 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 (PCL_A and PCL_B) to detect a conflict between two versions of the same messaging object.

Conflict Detection Algorithm

PCL_A **includes** PCL_B IFF for every **XID** in PCL_B there is an XID in PCL_A that has the same **NamespaceGuid** and same or greater **LocalId** part. The notation $PCL_A \succcurlyeq PCL_B$ will be used if PCL_A includes or is equal to PCL_B.

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 PCL_A includes PCL_B, then the version from replica A is newer and replaces the version in replica B.
2) If PCL_B includes or is equal to PCL_A, 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 MAY implement additional conflict detection mechanisms, as long as PCLs for object versions that do and do not conflict adhere to this criteria.

3.1.4.1.2 Resolution

At a minimum, servers MUST implement **conflict resolution** to the extent specified in this section. Servers MAY implement additional resolution algorithms. Any additional resolution algorithms MUST NOT result in the creation of conflict resolve **messages**, as specified in section 3.1.4.1.2.1.

A version that results from conflict resolution MUST have a PCL that makes it a *successor* of all conflicting versions. To achieve that, 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 IFF the **conflict detection** algorithm in 3.1.4.1.1 would determine that X is not in conflict and is newer than both A and B.

PCL_X is a **merge** of PCL_A and PCL_B IFF all of the following statements are true:

1) $PCL_X \subseteq (PCL_A \bigcup PCL_B)$
2) $PCL_X \succcurlyeq PCL_A$
3) $PCL_X \succcurlyeq PCL_B$

3.1.4.1.2.1 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. For more details about conflict resolve messages, see [MS-OXCSYNC] section 3.1.5.4. With the exceptions specified in [MS-OXCSYNC] section 3.1.5.4, 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 needed. The winner MUST be determined by the *last writer wins* algorithm, as specified in section 3.1.4.1.2.2. 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. This allows the other role to re-resolve the conflict during synchronization, while considering all (possibly, more than two) conflicting versions.

The last writer wins algorithm SHOULD be used for conflicts detected during **hierarchy synchronization** and **contents synchronization** of **normal messages**, unless specified otherwise in the **PidTagResolveMethod** property set on the **folder**.

3.1.4.1.2.2 Last Writer Wins Algorithm

The last writer wins algorithm uses the **PidTagLastModificationTime** property to determine the winning version, as specified in the following steps:

1) The version with the most recent **PidTagLastModificationTime** wins.
2) If the **PidTagLastModificationTime** value is equal on both objects, the winning version MAY <14> be determined by comparing byte-to-byte values of the **NamespaceGuid** field for **XIDs** in the **PidTagChangeKey** properties.
3) If the byte-to-byte comparison in step 2 determines that the **NamespaceGuid** fields are equal, the version being imported wins.

This algorithm SHOULD be used for conflicts detected during **hierarchy synchronization** and **contents synchronization** of **FAI** messages.

3.1.4.1.3 Reporting

Conflict reporting, if needed, 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 [MS-OXCSYNC] section 3.1.5.4.

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 its **attachments**, which clients MAY use to offer manual conflict resolution.

Determining whether to perform conflict reporting, and what method of conflict reporting SHOULD be used, is dependent on the operation that triggered the **conflict detection** and on the value of the **PidTagResolveMethod** property on the **folder**.

For example, **RopSynchronizationImportMessageChange** has a flag **FailOnConflict**, which switches between reporting by failing of the ROP and reporting by creating a conflict notification message. However, **RopSyhcnronizationImportHierarchyChange** MUST

detect and resolve, and MAY report, possible conflicts by using a conflict notification message.

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 discussed in sections 1.3 and 2.2.3 and their subsections. Otherwise, the client and server behavior remains undefined.

3.1.6 Creating Compact IDSETsOther Local Events

None.

3.2 Server Details

3.2.1 Abstract Data Model

3.2.1.1 Isolation of Download and Upload Operations

Clients MUST NOT assume that **upload** or **download** operations are 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**.

3.2.1.2 Creating Compact IDSETs

As the number of changes that happen to a **folder** grow over its lifetime, the sets of **MIDs** and **CNs** that need to be kept in **IDSETs** grow as well. The size of the IDSET is rarely a problem for **hierarchy synchronization**s due to the small number of folders commonly present in **mailboxes**. Therefore, this discussion focuses on **contents synchronization**. In this section, the term IDSET is used to refer to both IDSETs and **CNSETs**.

The following mechanisms are available to help optimize IDSETs for performance:

1. **IDSET compression**: The wire format of IDSETs is optimized for consecutive ranges and sets of non-consecutive 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 will be either deleted or moved to another folder, and so all of their IDs could be represented as one range. See section 3.3.1.1.1 for details.
3. **Collapsing of ranges**: If an IDSET is never iterated over and is only used in operations like "not in", it is possible to add ranges of IDs to the IDSET 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 (specified in section 3.2.4.1) only checks that a certain ID is not in the **PidTagCnset*** properties, it is possible to add CNs that were either never used or used on objects outside the synchronization scope to these IDSETs without affecting the outcome. Note that this MUST NOT be done for IDSETs that are ever iterated over, such as **PidTagIdsetGiven**, as it will change the outcome.

For example, an IDSET contains [10; 20] and [30; 40] for some **REPLGUID**. Because every internal **change number** within the same REPLGUID 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].

3.2.2 Timers

None.

3.2.3 Initialization

None.

3.2.4 Higher-Layer Triggered Events

3.2.4.1 Determining What Differences Need to be Downloaded

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 information about a change to an object if one of the following applies:
    - It is a **folder**
      AND a **change number** is not in **PidTagCnsetSeen**.
    - It is a **normal message**
      AND **SynchronizationFlag Normal** was set
      AND a change number is not in **PidTagCnsetSeen**.
    - It is an **FAI** message
      AND **SynchronizationFlag FAI** was set
      AND a change number is not in **PidTagCnsetSeenFAI**.
- If **SynchronizationFlag NoDeletions** is not set, include deletion information about objects that either:

- Have their internal identifiers present in **PidTagIdsetGiven** AND are missing from the **server replica**.
- Are folders that have never been reported as deleted.

- If **SynchronizationFlag NoSoftDeletions** is not set, include deletion information about objects that:
  - Have their internal identifiers present in **PidTagIdsetGiven** 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 **SynchronizationFlag ReadState** is set, include read state change information about messages that:
  - Do not have their change numbers for read and unread state in **PidTagCnsetRead** AND are not FAI messages AND have not had change information **downloaded** for them in this session.

The server needs to make sure that the **checkpoint ICS state** that is returned by **RopSynchronizationGetTransferState**, sent before the subsequent **RopFastTransferSourceGetBuffer**, 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 following table contains the nomenclature used to describe the invariants.

$Prop_{Index}$	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 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.
{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:

$$AllDeleted = (IdsetDeleted_D \cup IdsetSoftDeleted_D \cup IdsetExpired_D)$$

$$IdsetGiven_C = (IdsetGiven_I \cup \{change.Id\}) \setminus AllDeleted$$

$$CnsetSeen_C = CnsetSeen_I \cup \{change_{Normal}.Cn\}$$

$$CnsetSeenFAI_C = CnsetSeenFAI_I \cup \{change_{FAI}.Cn\}$$

$$CnsetRead_C = CnsetRead_I \cup \{readStateChange.ReadCn\}$$

Invariants for $CnsetSeen_C$, $CnsetSeenFAI_C$, and $CnsetRead_C$ are amended in 3.2.1.2.

$$IdsetGiven_I \supseteq \{changes_{Partial}.Id\}$$

$$IdsetGiven_I \supseteq (IdsetRead_D \cup IdsetUnread_D)$$

$$\{readStateChange.Id\} = IdsetRead_D \cup IdsetUnread_D$$

$$\{change.Id\} \cap AllDeleted = \varnothing$$

$$\{change.Cn\} \cap (CnsetSeen_I \cup CnsetSeenFAI_I) = \varnothing$$

$$\{readStateChange.Id\} \cap AllDeleted = \varnothing$$

$$\{readStateChange.Id\} \cap \{change.Id\} = \varnothing$$

3.2.4.2 Generating the PidTagSourceKey Value

When the **PidTagSourceKey** value is missing, the server MUST generate it by producing a **GID** from the internal identifier (**MID** or **FID**) of the object by using the same mapping algorithm as described for **RopLongTermIdFromId** (as specified in [MS-OXCSTOR]).

The only exception is when a server needs to generate this **property** on the fly for a **folder**, which is a root of the current **hierarchy synchronization** download operation (that is, it is the folder that was passed to **RopSynchronizationConfigure**). In this case, **PidTagSourceKey** MUST be output as a zero-length PtypBinary.

3.2.4.3 Read State Change Tracking

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, a separate **change number** on the message, the *read state change number*, SHOULD be assigned a new value. 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 change to be efficiently **downloaded** to a client as the **MID** in an IDSET **PidTagIdsetRead** or **PidTagIdsetUnread**, compressed together with read state changes to other messages in the **synchronization scope**.

3.2.4.4 Fast Transfer Copy Operations

3.2.4.4.1 Download

When producing **FastTransfer stream**s 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 an object to be included in the output FastTransfer stream. If a permission check for an object fails, the **PidTagEcWarning** meta-property SHOULD be output in a FastTransfer stream, wherever allowed by its syntactical structure, to signal a client about incomplete content.

3.2.4.4.1.1 Receiving a RopFastTransferSourceGetBuffer

Servers SHOULD fail any successive calls to **RopFastTransferSourceGetBuffer**, after the previous iteration returns a buffer with a **ReturnValue** other that **Success** or **ServerBusy**.

3.2.4.5 Incremental Change Synchronization

3.2.4.5.1 Downloading State

3.2.4.5.1.1 Receiving a RopSynchronizationGetTransferState

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.

3.2.4.5.2 Upload

3.2.4.5.2.1 Receiving a RopSynchronizationImportMessageChange

Upon successful completion of the **RopSynchronizationImportMessageChange ROP**, the **ICS state** on the synchronization context MUST be updated to include a new **change**

number in either the **PidTagCnsetSeen** or **PidTagCnsetSeenFAI** property, depending on whether a particular **message** is a **normal message** or an **FAI** message.

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 **RopSaveChangesMessages** is called.

3.2.4.5.2.2 Receiving a RopSynchronizationImportHierarchyChange

Upon successful completion of this **ROP**, the **ICS state** on the synchronization context MUST be updated to include a new **change number** in the **PidTagCnsetSeen** property.

If a conflict has occurred, the server:

- SHOULD NOT update the **PidTagCnsetSeen** property, and let the clients **download** a result of **conflict resolution**.
- MAY generate a conflict notification message. See section 3.1.4.1.3 for more details.
- MUST return **Success** in the **ReturnValue**.

The server MUST ignore the properties in **PropertyValues**, which are also present in **HierarchyValues**.

3.2.4.5.2.3 Receiving a RopSynchronizationImportMessageMove

Upon successful completion, the **ICS state** on the synchronization context MUST be updated to include **change number**s of **messages** in the destination **folder** in either the **PidTagCnsetSeen** or **PidTagCnsetSeenFAI** property, depending on whether a message is a **normal message** or an **FAI** message.

3.2.4.5.2.4 Receiving a RopSynchronizationImportDeletes

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 **RopSynchronizationImportHierarchyChange** or **RopSynchronizationImportMessageChange** from restoring them back.

The protocol does not dictate that deletions of all objects passed in the request to this **ROP** MUST happen in a transacted way. However, to minimize the possibility of putting replicas into a desynchronized state and because a protocol does not let clients know in any way what part of an operation has succeeded, servers SHOULD make a reasonable effort to predict whether all deletions will succeed, and if a deletion will not succeed, report a failure right away, instead of partially completing an operation.

3.2.4.5.2.5 Receiving a RopSynchronizationImportReadStateChanges

The **RopSynchronizationImportReadStateChanges** ROP is a batch variant of **RopSetMessageReadFlag**, which also takes care of updating the **ICS state**. The net effect of

changing the read state **message** by message by using **RopSetMessageReadFlag** MUST be identical to changing the read state in bulk by using **RopSynchronizationImportReadStateChanges**.

Requests to change the read state of **FAI** messages MUST be ignored. Upon successful completion, the ICS state on the synchronization context MUST be updated by adding the new **change number** to the **PidTagCnsetRead** property.

The protocol does not dictate that the change of the read state for all objects passed in the **ROP** request MUST happen in a transacted way. However, to minimize the possibility of putting replicas into a desynchronized state and because a protocol does not let clients know what part of an operation has succeeded, servers SHOULD make a reasonable effort to predict whether changes of state for all messages will succeed, and if the changes of state will not succeed, report a failure immediately, instead of partially completing an operation.

3.2.4.5.2.6 Receiving a RopGetLocalReplicaIds

A server MAY 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 MAY 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 3.3.4.2.2.7.

3.2.4.5.2.7 Receiving a RopSetLocalReplicaMidsetDeleted

A server MAY add ranges of IDs supplied through this **ROP** to the **deleted item list**, if one is maintained for the **folder**. One possible reason for doing that is to be able to compress the deleted item list by using the IDSET optimization algorithm specified in section 3.2.1.2.

A server MAY ensure that ranges supplied as request fields to this ROP are allocated by using **RopGetLocalReplicaIds**.

3.2.4.6 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 property A in the **PropertyTags** field of the request buffer of a **RopFastTransferSourceCopyProperties ROP** that is configured with an **Attachment object** as an **InputServerObject** will affect the set of properties to be copied for this attachment, but not its **embedded message** or any **attachments** that it might contain.
- Specifying the **PidTagFolderAssociatedContents** property in the **PropertyTags** field of the request buffer of a **RopFastTransferSourceCopyTo** ROP that is configured with a **Folder object** as an **InputServerObject** will only exclude **FAI**

Message objects from copying this specific **folder**, but not any of its descendant folders.
- Specifying the **PidTagMessageRecipients** property in the **PropertyTags** fields of the request buffer of a **RopSynchronizationConfigure** ROP will exclude **recipient subobjects** from all **message** changes downloaded in that operation, but it will 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. See section 3.2.4.8 for more details.

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 **CopyFlag CopySubfolders** flag (as specified in section 2.2.3.1.1.1.1) includes all subfolders of the current folder into the operation scope.
- Specifying **CopyFlag SendEntryId** flag includes all identification properties for all objects being downloaded.

Whenever **subobject** filters have an effect, servers MUST output a **PidTagFXDelProp** meta-property 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 **meta-property PidTagMessageRecipients** in the **PropertyTags** field of the request buffer of the **RopFastTransferSourceCopyProperties** ROP that is configured with a **Message** object as an **InputServerObject**, will direct the server to output **PidTagFXDelProp PidTagMessageRecipients** 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 **PidTagFXDelProp** meta-property MUST NOT be output, or are filtered in; that is, they MUST be output one after another, prefixed by the **PidTagFXDelProp** meta-property.

3.2.4.7 Properties to Ignore on Upload

Unless specified otherwise in **property list restriction tables**, properties that belong to the **provider-defined internal non-transmittable** range, as specified in [MS-OXPROPS] section 1.3.3, MUST be ignored on **upload**.

3.2.4.8 Properties to Ignore on Download

Unless specified otherwise in **property list restriction tables**, **propValue** elements of **FastTransfer streams** that belong to the provider-defined internal non-transmittable range (as specified in [MS-OXPROPS] section 1.3.3) MUST be excluded from **download**.

3.2.5 Timer Events

None.

3.2.6 Other Local Events

None.

3.3 Client Details

This section provides client-specific details related to bulk data transfer. The Mailbox Synchronization Protocol Specification [MS-OXCSYNC] also contains important client-specific details related to bulk data transfer.

3.3.1 Abstract Data Model

3.3.1.1 Object and Change Identification

The following three alternative mechanisms are available to clients that need 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.

3.3.1.1.1 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**. 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 **RopSyncrhonizationImportHierarchyChange** (as specified in section 2.2.3.2.4.3) or **RopSyncrhonizationImportMessageChange** (as specified in section 2.2.3.2.4.2). Note that these IDs MUST NOT be used for **change numbers**.

Clients MUST generate **foreign identifiers** to identify changes to objects in the local replica, as specified in section 3.3.1.1.3.

This mechanism is being serviced by two ROPs, **RopGetLocalReplicaIds** (as specified in section 2.2.3.2.4.7) and **RopSetLocalReplicaMidsetDeleted** (as specified in section 2.2.3.2.4.8).

To help compression of IDSETs 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. One possible mechanism to achieve this is to allocate 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 **RopSynchronizationImportDeletes**.

Clients MUST report ranges of server-allocated IDs, which will never be used for any messages in a folder, through **RopSetLocalReplicaMidsetDeleted**. For an example, see section 3.3.4.2.2.8.

3.3.1.1.2 Use Online Mode ROPs

In this approach, clients MUST **upload** objects created in their **local replica** by using the regular, non-synchronization **ROPs**, such as **RopCreateFolder** or **RopCreateMessage**, as specified in [MS-OXCMSG] and [MS-OXCFOLD], 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 **PidTagLastModificationTime**.
- Clients are entirely responsible for updating the **ICS state** to prevent uploaded objects from being **downloaded** during a subsequent synchronization download operation.

3.3.1.1.3 Foreign Identifiers

Clients MUST generate **foreign identifiers** to identify changes to objects in the local replica. Foreign identifiers are represented as **XIDs** and MUST NOT have the same byte length as **GIDs**; that is, the number of bytes in the **LocalId** field that follows a **NamespaceGuid** in the XID structure MUST be different from the size of **GLOBCNT**, which is 6 bytes. At the same time, foreign identifiers that share the same **NamespaceGuid** MUST have the same length of the **LocalId** part.

Clients MUST create foreign identifiers within the **NamespaceGuids** they generated, and MUST NOT use any **REPLGUIDs** 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 3.1.4.1.1.

3.3.1.2 Synchronization Scope

To be able to perform an **Incremental Change Synchronization (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 **RopSynchronizationConfigure**, as specified in section 2.2.3.2.1.1). **Synchronization scope** is determined by using the following variables:

- Mailbox
- Synchronization type (**hierarchy** or **contents**)

- **Folder** within the mailbox
- Restrictions on **messages** within the folder that are included in the scope (for **contents synchronization** only)

Synchronization for each of the scopes can be performed independently. For each synchronization scope, a client MUST persist the corresponding **ICS state**, and be sure to 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 **SynchronizationFlag** field of **RopSynchronizationConfigure**. 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:

1. 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 will produce the same end 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** equals **FALSE**}) => (diff1, icsStateA)
2. IcsDownload(icsStateA, Normal | FAI, {**PidTagAssociated** equals **TRUE**}) => (diff2, icsStateB)

As a result, this sequence will not yield the same result:

- diff1 will contain soft-deletion notifications for any previously **downloaded** messaging objects mentioned in icsStateX.**PidTagIdsetGiven**, which do not have **PidTagAssociated** equals **FALSE**.

- diff2 will contain soft-deletions for all messaging objects mentioned in icsStateA.**PidTagIdsetGiven**.
- icsStateB.**PidTagIdsetGiven** will only contain IDs of FAI messages.

3.3.2 Timers

None.

3.3.3 Initialization

None.

3.3.4 Higher-Layer Triggered Events

None.

3.3.4.1 Fast Transfer Copy Operations

3.3.4.1.1 Download

3.3.4.1.1.1 Sending a RopFastTransferSourceGetBuffer

The **FastTransfer stream** on **download** is read-only and non-seekable, and is usually generated on-the-fly. Once it is obtained, data cannot be re-queried, unless the operation is re-configured 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 MAY decode portions of the FastTransfer stream as they arrive in **RopFastTransferSourceGetBuffer** response buffers, and then query for more when they need to.

3.3.4.1.1.2 Sending a RopTellVersion

Clients MUST pass the version exactly as it was obtained from the **EcDoConnect** or **EcDoConnectEx** call results. For more details about the only application scenario for this **ROP**, server-to-client-to-server **upload**, see section 3.3.4.1.2.1.

3.3.4.1.2 Upload

3.3.4.1.2.1 Server-to-Client-to-Server Upload

To optimize copying **messaging objects** between two different **mailboxes** on two different servers by using FastTransfer **download** paired with FastTransfer download, a client MAY specify the **ForUpload** flag in **SendOptions**, which instructs the source server to produce a **FastTransfer stream** that is optimized for the destination server.

Clients MUST NOT parse the FastTransfer stream produced by the source server, as it MAY contain any kind of 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*** requests to server A to configure a FastTransfer download context, while setting the **ForUpload** flag in the **SendOptions** field.
2. Send the **RopFastTransferDestinationConfigure** request to server B to configure a FastTransfer **upload** context.
3. Send the **RopTellVersion** request on a FastTransfer download context with a version of server B.
4. Send the **RopTellVersion** request on a FastTransfer upload context with a version of server A.
5. Iteratively send **RopFastTransferSourceGetBuffer** requests on a FastTransfer download context followed by **RopFastTransferDestinationPutBuffer** requests on a FastTransfer upload context until there is no more data.
6. Release both FastTransfer contexts.

3.3.4.2 Incremental Change Synchronization

3.3.4.2.1 Downloading State

3.3.4.2.1.1 Sending a RopSynchronizationGetTransferState

Clients only need to use the **RopSynchronizationGetTransferState** ROP when performing synchronization **uploads**, as it is the only way to obtain the **ICS state** maintained on the **synchronization upload context**. For synchronization **downloads**, the **final ICS state** is downloaded at the end of the **FastTransfer stream**, and this **ROP** can only be used to obtain the **checkpoint ICS state**, as an alternative to using client-side checkpointing (as specified in [MS-OXCSYNC] section 3.1.5.3.9.1).

3.3.4.2.2 Upload

Clients MAY <15> perform a synchronization **upload** without uploading the **initial ICS state** properties into a **synchronization upload context**, because the behavior of the **RopSynchronizationImport* ROPs** does not depend on the initial **ICS state**. In that case, a server MAY **download** the changes uploaded in this session during the subsequent ICS download.

3.3.4.2.2.1 Sending a RopSynchronizationOpenCollector

Be sure to update the stored **PidTagIdsetGiven** value with internal identifiers of the objects that were imported into the **server replica**. These identifiers are either returned in the responses of **RopSynchronizationImport* ROPs**, or can be extracted from **GIDs** sent as input **PidTagSourceKey** values.

3.3.4.2.2.2 Sending a RopSynchronizationImportMessageChange

When **uploading** new **messages**, clients SHOULD add their **MIDs** to the **PidTagIdsetGiven** value upon successful completion of this **ROP**.

Note that because a server returns an empty message from **RopSynchronizationImportMessageChange**, even when uploading changes to an existing message, this ROP can only be used to perform upload of full message changes or new messages. If a client wants to upload partial message changes, it SHOULD take them outside the synchronization upload operation, by initiating an upload by using **RopOpenMessage** followed by other ROPs discussed in [MS-OXCMSG], such as **RopSetProperties** and **RopFlushRecipients**. However, these ROPs do not let the client set values to any of the properties that **RopSynchronizationImportMessageChange** accepts.

3.3.4.2.2.3 Sending a RopSynchronizationImportHierarchyChange

When **uploading** new **folders**, clients SHOULD update the **ICS state** that corresponds to the chosen **synchronization scope** by adding **FIDs** of new folders to the **PidTagIdsetGiven property** upon successful completion of this **ROP**.

3.3.4.2.2.4 Sending a RopSynchronizationImportMessageMove

When **uploading** new **messages**, clients SHOULD update the **ICS state** of the source **folder** by removing **MIDs** of moved messages from its **PidTagIdsetGiven** property. Otherwise, the client MUST be prepared to receive deletion notifications for these messages in the source folder during the next ICS **download**.

3.3.4.2.2.5 Sending a RopSynchronizationImportDeletes

Clients SHOULD update the **ICS state** of the chosen **synchronization scope** by removing internal identifiers of deleted objects from its **PidTagIdsetGiven** property. Otherwise, clients MUST be prepared to 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 fail, 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. Clients that use this ROP SHOULD have a strategy to retry this operation, which MAY 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, like **RopDeleteFolder** and **RopDeleteMessages**.<16>
4) Perform the ICS download, resolving server changes against their own pending **upload**.
5) Skip an object and undo the operation in the **local replica**.

3.3.4.2.2.6 Sending a RopSynchronizationImportReadStateChanges

Clients SHOULD expect this **ROP** to fail if any 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 MAY 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 **RopSetMessageReadFlag**.
4) Perform the **ICS** download, resolving server changes against their own pending **upload**.
5) Skip an object and undo the operation in the **local replica**.

3.3.4.2.2.7 Sending a RopGetLocalReplicaIds

Clients SHOULD NOT allocate another batch of IDs until the one they allocated before is used up or near depletion. Allocating IDs in batches of moderate size, between 0x00000200 and 0x0000FFFF, is recommended. Note that servers SHOULD impose restrictions on the number of IDs that can be allocated at one time.

3.3.4.2.2.8 Sending a RopSetLocalReplicaMidsetDeleted

The following example shows a possible implementation of the client with regards to assignment of server-allocated IDs (section 3.3.1.1.1) 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 **RopSetLocalReplicaMidsetDeleted**.

1. Initially, a client has no server-allocated IDs that it can assign to objects that are created when working offline, so it needs to ask a server to allocate a block of IDs by sending **RopGetLocalReplicaIds**. The server responds with a block of IDs that the client stores in a local replica.
2. The client needs 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 **RopGetLocalReplicaIds** with a folder, so that IDs from that range can be used for new or moved items in that folder.
3. If a folder does not have a range of server-allocated IDs associated with it, because the previous range was depleted (say, [A; B]), the client would have to allocate another range (say, [C; D]) from the block obtained in step 1 and associate it with that folder.
4. After a new range [C; D] is associated with a folder, the client knows that all ids in [B+1; C-1] will never be used in that folder, because they have already been associated with other folders. Therefore, the client can send **RopSetLocalReplicaMidsetDeleted** for that folder with the [B+1; C-1] range.

3.3.5 Message Processing Events and Sequencing Rules

3.3.6 Timer Events

None.

3.3.7 Other Local Events

None.

4 Protocol Examples

4.1 IDSET Serialization

To efficiently transfer large numbers of **MIDs** and **FIDs** that identify changed or new **messaging objects**, the MIDs and FIDs are serialized into an IDSET for transfer across the wire. The following example shows how to format and serialize an IDSET. Because of the variability of the **GLOBSET** 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. The following is just one way to encode an IDSET.

This example uses an IDSET with following four MID values:

IDSET

	Value	**REPLID**	**GLOBCNT**
MID1	01 00 00 00 00 00 00 05	0001	000000000005
MID2	01 00 00 00 00 00 00 06	0001	000000000006
MID3	01 00 00 00 00 00 00 10	0001	000000000010
MID4	02 00 00 00 00 00 00 09	0002	000000000009

The IDSET MUST first be properly formatted for serializations. See section 3.1.1.3.1 for more details about how to format an IDSET.

The following diagram represents how the IDSET MUST 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.

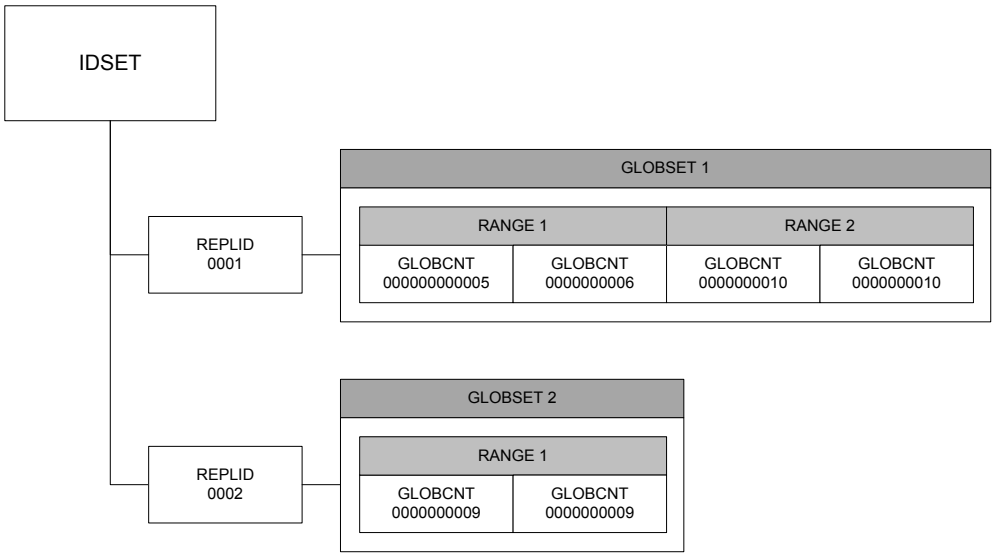


Figure 3: Sample IDSET used in the serialization example

This example serializes the IDSET by using the REPLID format. See section 2.2.2.3 for more details about the different serialization formats of an IDSET.

For each REPLID/GLOBSET pair, the REPLID MUST be added to the serialization buffer before the encoded GLOBSET. They MUST be ordered based on the REPLID value where they are ordered from lowest to higheset value.

The serialization buffer will resemble 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 MUST 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 MUST 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 MUST 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 will be put into a **Bitmask** command, either the **Bitmask** command or the **Range** command could be used. Because they both will 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 will 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 will only contain 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 will generate 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 needed 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 MUST 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 will fill the common array, it will generate two identical GLOBCNT values when decoded, producing a singleton GLOBCNT range.

Current Encoding Buffer
06 00 00 00 00 00 09

We now have encodings in the encoding buffer 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.

4.2 FastTransfer Stream Produced by Contents Synchronization Download

The following example shows the sample output of a **FastTransfer stream** that is downloaded to a client during a **contents synchronization**. The download operation was configured by using the **RopSynchronizationConfigure** command with the following fields specified in the request buffer:

Field of the request buffer	Value
SynchronizationType	Contents
SendOptions	Unicode, RecoverMode, ForceUnicode, PartialItem <4>
SynchronizationFlags	Unicode, ReadState, FAI, Normal, NoForeignIdentifiers, BestBody, Progress
RestrictionDataSize	0
RestrictionData	< missing >
SynchronizationExtraFlags	Eid, Cn, OrderByDeliveryTime

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 logial relationship of the data delimited by the markers.

IncrSyncProgressMode

IncrSyncProgressPerMsg

IncrSyncChg

IncrSyncMsg

StartRecip

EndToRecip

NewAttach

StartEmbed

StartRecip

EndToRecip

EndEmbed

EndAttach

IncrSyncDel

IncrSyncRead

IncrSyncStateBegin

IncrSyncStateEnd

IncrSyncEnd

Bytes on the wire	**Description**
0B 00 74 40	**marker** **IncrSyncProgressMode (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 (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 (40120003 [Int32])**
02 01 E0 65	propDef PidTagSourceKey (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** **00 00 00 78-2E 21**	varSizeValue ....A ....c..f ...x.!
40 00 08 30	propDef PidTagLastModificationTime (30080040 [SysTime])
FC 65 69 CF-C0 84 C8 01	fixedSizeValue [SysTime] 2008-03-13T04:15:02.8437500
02 01 E2 65	propDef PidTagChangeKey (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 (65E30102 [Binary])
17 00 00 00	length 23 (0x17)
	varSizeValue

Bytes on the wire	**Description**
16 19 D7 FB-0F 06 16 A1 **41 BF F6 91-C7 63 DA A8** **66 00 00 00-78 4D 1C**	.... A....c.. f...xM.
0B 00 AA 67	propDef PidTagAssociated (67AA000B [Bool])
00 00	fixedSizeValue [Bool] False
14 00 4A 67	propDef PidTagMid (674A0014 [Int64])
01 00 00 00-00 78 2E 21	fixedSizeValue [Int64] 2390980393575645185
14 00 A4 67	propDef PidTagChangeNumber (67A40014 [Int64])
01 00 00 00-00 78 4D 1C	fixedSizeValue [Int64] 2039418147664035841
03 00 15 40	**marker** **IncrSyncMsg (40150003 [Int32])**
0B 00 02 00	propDef PidTagAlternateRecipientAllowed (0002000B [Bool])
01 00	fixedSizeValue [Bool] True
03 00 17 00	propDef PidTagImportance (00170003 [Int32])
01 00 00 00	fixedSizeValue [Int32] 1
1F 00 1A 00	propDef PidTagMessageClass (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 (0023000B [Bool])
00 00	fixedSizeValue [Bool] False
03 00 26 00	propDef PidTagPriority (00260003 [Int32])
00 00 00 00	fixedSizeValue [Int32] 0
0B 00 29 00	propDef PidTagReadReceiptRequested (0029000B [Bool])
00 00	fixedSizeValue [Bool] False
03 00 36 00	propDef PidTagSensitivity (00360003 [Int32])
00 00 00 00	fixedSizeValue [Int32] 0
1F 00 37 00	propDef PidTagSubject (0037001F [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 ...
40 00 39 00	propDef PidTagClientSubmitTime (00390040 [SysTime])
	fixedSizeValue

Bytes on the wire	**Description**
80 BA A7 B7-BC 84 C8 01	[SysTime] 2008-03-13T03:45:45.0000000
02 01 3B 00	propDef PidTagSentRepresentingSearchKey (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 (003D001F [Unicode])
02 00 00 00	length 2 (0x2)
00 00	varSizeValue ..
02 01 3F 00	propDef PidTagReceivedByEntryId (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 (0040001F [Unicode])
06 00 00 00	length 6 (0x6)
74 00 31 00-00 00	varSizeValue t.1...
02 01 41 00	propDef PidTagSentRepresentingEntryId (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 (0042001F [Unicode])
06 00 00 00	length 6 (0x6)
74 00 31 00-00 00	varSizeValue t.1...
02 01 43 00	propDef PidTagReceivedRepresentingEntryId (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 ...	

Bytes on the wire	**Description**
1F 00 44 00	propDef PidTagReceivedRepresentingName (0044001F [Unicode])
06 00 00 00	length 6 (0x6)
74 00 31 00-00 00	varSizeValue t.1...
02 01 51 00	propDef PidTagReceivedBySearchKey (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 (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 ...	
1F 00 64 00	propDef PidTagSentRepresentingAddressType (0064001F [Unicode])
06 00 00 00	length 6 (0x6)
45 00 58 00-00 00	varSizeValue E.X...
1F 00 65 00	propDef PidTagSentRepresentingEmailAddress (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 (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 (00710102 [Binary])
16 00 00 00	length 22 (0x16)
01 C8 84 BC-B6 CB 8A CC 1E B8 32 77-43 2B A1 C6	varSizeValue ....2wC+..

Bytes on the wire	**Description**
83 9A 4A F4-BC 14	..J...
1F 00 75 00	propDef PidTagReceivedByAddressType (0075001F [Unicode])
06 00 00 00	length 6 (0x6)
45 00 58 00-00 00	varSizeValue E.X...
1F 00 76 00	propDef PidTagReceivedByEmailAddress (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 (0077001F [Unicode])
06 00 00 00	length 6 (0x6)
45 00 58 00-00 00	varSizeValue E.X...
1F 00 78 00	propDef PidTagReceivedRepresentingEmailAddress (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 (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 (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 (0C190102 [Binary])
79 00 00 00	length 121 (0x79)
	varSizeValue

Bytes on the wire	**Description**
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	......@. .B...... +/...... ..../O=F IRST ORG
	... value truncated ...
1F 00 1A 0C	propDef PidTagSenderName (0C1A001F [Unicode])
06 00 00 00	length 6 (0x6)
74 00 31 00-00 00	varSizeValue t.1...
02 01 1D 0C	propDef PidTagSenderSearchKey (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 (0C1E001F [Unicode])
06 00 00 00	length 6 (0x6)
45 00 58 00-00 00	varSizeValue E.X...
1F 00 1F 0C	propDef PidTagSenderEmailAddress (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 ...
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 (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 (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 (0x8124 [PSETID_Task]) [Bool]
00 00	fixedSizeValue [Bool] False
40 00 06 0E	propDef PidTagMessageDeliveryTime (0E060040 [SysTime])

Bytes on the wire	**Description**
80 E7 D8 B8-BC 84 C8 01	fixedSizeValue [SysTime] 2008-03-13T03:45:47.0000000
03 00 07 0E	propDef PidTagMessageFlags (0E070003 [Int32])
31 00 00 00	fixedSizeValue [Int32] 49
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 (0x8129 [PSETID_Task]) [Int32]
00 00 00 00	fixedSizeValue [Int32] 0
03 00 17 0E	propDef PidTagMessageStatus (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 (0x8111 [PSETID_Task]) [Int32]
00 00 00 00	fixedSizeValue [Int32] 0
1F 00 1D 0E	propDef PidTagNormalizedSubject (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 (0E1F000B [Bool])
01 00	fixedSizeValue [Bool] True
03 00 23 0E	propDef PidTagInternetArticleNumber (0E230003 [Int32])
26 00 00 00	fixedSizeValue [Int32] 38
03 00 79 0E	propDef PidTagTrustSender (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 (0x8110 [PSETID_Task]) [Int32]
00 00 00 00	fixedSizeValue [Int32] 0
03 00 F7 0F	propDef PidTagAccessLevel (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 (0x8121 [PSETID_Task]) [Unicode]

Bytes on the wire	**Description**
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 (0x8123 [PSETID_Task]) [Int32]
FF FF FF 7F	fixedSizeValue [Int32] 2147483647
1F 00 35 10	propDef PidTagInternetMessageId (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 (10800003 [Int32])
FF FF FF FF	fixedSizeValue [Int32] -1
40 00 07 30	propDef PidTagCreationTime (30070040 [SysTime])
A2 DA EF B9-BC 84 C8 01	fixedSizeValue [SysTime] 2008-03-13T03:45:48.8281250
40 00 08 30	propDef PidTagLastModificationTime (30080040 [SysTime])
FC 65 69 CF-C0 84 C8 01	fixedSizeValue [SysTime] 2008-03-13T04:15:02.8437500
02 01 0B 30	propDef PidTagSearchKey (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 (3A40000B [Bool])
01 00	fixedSizeValue [Bool] True
03 00 DE 3F	propDef PidTagInternetCodepage (3FDE0003 [Int32])
9F 4E 00 00	fixedSizeValue [Int32] 20127
03 00 F1 3F	propDef PidTagMessageLocaleId (3FF10003 [Int32])
09 04 00 00	fixedSizeValue [Int32] 1033
03 00 FD 3F	propDef PidTagMessageCodepage (3FFD0003 [Int32])
E3 04 00 00	fixedSizeValue [Int32] 1251
03 00 19 40	propDef PidTagSenderFlags (40190003 [Int32])
00 00 00 00	fixedSizeValue [Int32] 0
03 00 1A 40	propDef PidTagSentRepresentingFlags (401A0003 [Int32])

Bytes on the wire	**Description**
00 00 00 00	fixedSizeValue [Int32] 0
03 00 1B 40	propDef PidTagReceivedByFlags (401B0003 [Int32])
00 00 00 00	fixedSizeValue [Int32] 0
03 00 1C 40	propDef PidTagReceivedRepresentingFlags (401C0003 [Int32])
00 00 00 00	fixedSizeValue [Int32] 0
03 00 76 40	propDef PidTagContentFilterSpamConfidenceLevel (40760003 [Int32])
FF FF FF FF	fixedSizeValue [Int32] -1
03 00 02 59	propDef PidTagInternetMailOverrideFormat (59020003 [Int32])
00 00 16 00	fixedSizeValue [Int32] 1441792
03 00 09 59	propDef PidTagMessageEditorFormat (59090003 [Int32])
02 00 00 00	fixedSizeValue [Int32] 2
03 00 C6 65	propDef PidTagSecureSubmitFlags (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 (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 (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 (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 (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 00 00 00 46-00 54 85 00 00	propDef PidLidCurrentVersionName (0x8554 [PSETID_Common]) [Unicode]
0A 00 00 00	length 10 (0xA)

Bytes on the wire	**Description**
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 (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 (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 (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 79 00 00 00	varSizeValue B.l.u.e. .C.a.t. e.g.o.r. y...
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 (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 (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 (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	propDef PidLidInternetAccountName (0x8580 [PSETID_Common]) [Unicode]

Bytes on the wire	**Description**
00	
26 00 00 00	length 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 (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 (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 00	propDef PidLidReminderDelta (0x8501 [PSETID_Common]) [Int32]
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 (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 (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 (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 (0x8113 [PSETID_Task]) [Int32]

Bytes on the wire	**Description**
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 (0x8112 [PSETID_Task]) [Int32]
01 00 00 00	fixedSizeValue [Int32] 1
02 01 13 10	propDef PidTagBodyHtml (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 PidTagFXDelProp (40160003 [Int32])
0D 00 12 0E	fixedSizeValue PidTagMessageRecipients (0E12000D [Object])
03 00 03 40	**marker** **StartRecip (40030003 [Int32])**
03 00 00 30	propDef PidTagRowid (30000003 [Int32])
00 00 00 00	fixedSizeValue [Int32] 0
1F 00 02 30	propDef PidTagAddressType (3002001F [Unicode])
06 00 00 00	length 6 (0x6)
45 00 58 00-00 00	varSizeValue E.X...
1F 00 03 30	propDef PidTagEmailAddress (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 00-49 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 01 30	propDef PidTagDisplayName (3001001F [Unicode])
06 00 00 00	length 6 (0x6)
74 00 31 00-00 00	varSizeValue t.1...
02 01 F6 0F	propDef PidTagInstanceKey (0FF60102 [Binary])
04 00 00 00	length 4 (0x4)
00 00 00 00	varSizeValue ....
03 00 15 0C	propDef PidTagRecipientType (0C150003 [Int32])
01 00 00 00	fixedSizeValue [Int32] 1

Bytes on the wire	**Description**
02 01 FF 0F	propDef PidTagEntryId (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 (300B0102 [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
... value truncated ...	
1F 00 20 3A	propDef PidTagTransmittableDisplayName (3A20001F [Unicode])
06 00 00 00	length 6 (0x6)
74 00 31 00-00 00	varSizeValue t.1...
0B 00 0F 0E	propDef PidTagResponsibility (0E0F000B [Bool])
01 00	fixedSizeValue [Bool] True
0B 00 40 3A	propDef PidTagSendRichInfo (3A40000B [Bool])
01 00	fixedSizeValue [Bool] True
03 00 FD 5F	propDef PidTagRecipientFlags (5FFD0003 [Int32])
01 00 00 00	fixedSizeValue [Int32] 1
02 01 F7 5F	propDef PidTagRecipientEntryId (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 ...	
1F 00 FE 39	propDef PidTagPrimarySmtpAddress (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 (39050003 [Int32])

Bytes on the wire	**Description**
00 00 00 40	fixedSizeValue [Int32] 1073741824
03 00 00 39	propDef PidTagDisplayType (39000003 [Int32])
00 00 00 00	fixedSizeValue [Int32] 0
03 00 FE 0F	propDef PidTagObjectType (0FFE0003 [Int32])
06 00 00 00	fixedSizeValue [Int32] 6
1F 00 FF 39	propDef PidTag7BitDisplayName (39FF001F [Unicode])
06 00 00 00	length 6 (0x6)
74 00 31 00-00 00	varSizeValue t.1...
1F 00 00 3A	propDef PidTagAccount (3A00001F [Unicode])
06 00 00 00	length 6 (0x6)
74 00 31 00-00 00	varSizeValue t.1...
03 00 FF 5F	propDef PidTagRecipientTrackStatus (5FFF0003 [Int32])
00 00 00 00	fixedSizeValue [Int32] 0
03 00 DE 5F	propDef PidTagRecipientResourceState (5FDE0003 [Int32])
00 00 00 00	fixedSizeValue [Int32] 0
1F 00 F6 5F	propDef PidTagRecipientDisplayName (5FF6001F [Unicode])
06 00 00 00	length 6 (0x6)
74 00 31 00-00 00	varSizeValue t.1...
03 00 DF 5F	propDef PidTagRecipientOrder (5FDF0003 [Int32])
00 00 00 00	fixedSizeValue [Int32] 0
03 00 04 40	**marker** **EndToRecip (40040003 [Int32])**
03 00 16 40	propDef PidTagFXDelProp (40160003 [Int32])
0D 00 13 0E	marker PidTagMessageAttachments (0E13000D [Object])
03 00 00 40	**marker** **NewAttach (40000003 [Int32])**
03 00 21 0E	propDef PidTagAttachNumber (0E210003 [Int32])
00 00 00 00	fixedSizeValue [Int32] 0
02 01 02 37	propDef PidTagAttachEncoding (37020102 [Binary])
00 00 00 00	length 0 (0x0)
03 00 0B 37	propDef PidTagRenderingPosition (370B0003 [Int32])
FF FF FF FF	fixedSizeValue [Int32] -1
03 00 20 0E	propDef PidTagAttachSize (0E200003 [Int32])
	fixedSizeValue

Bytes on the wire	**Description**
E7 15 00 00	[Int32] 5607
03 00 F7 0F	propDef PidTagAccessLevel (0FF70003 [Int32])
00 00 00 00	fixedSizeValue [Int32] 0
40 00 07 30	propDef PidTagCreationTime (30070040 [SysTime])
E2 EA E3 B1-BC 84 C8 01	fixedSizeValue [SysTime] 2008-03-13T03:45:35.3281250
40 00 08 30	propDef PidTagLastModificationTime (30080040 [SysTime])
E2 EA E3 B1-BC 84 C8 01	fixedSizeValue [SysTime] 2008-03-13T03:45:35.3281250
03 00 05 37	propDef PidTagAttachMethod (37050003 [Int32])
05 00 00 00	fixedSizeValue [Int32] 5
02 01 09 37	propDef PidTagAttachRendering (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 (37140003 [Int32])
00 00 00 00	fixedSizeValue [Int32] 0
0B 00 FE 7F	propDef PidTagAttachmentHidden (7FFE000B [Bool])
00 00	fixedSizeValue [Bool] False
1F 00 04 37	propDef PidTagAttachFilename (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 (7FFF000B [Bool])
00 00	fixedSizeValue [Bool] False
1F 00 01 30	propDef PidTagDisplayName (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 (0FF90102 [Binary])
04 00 00 00	length 4 (0x4)
00 00 00 00	varSizeValue ....
03 00 01 40	**marker** **StartEmbed (40010003 [Int32])**
	propDef

Bytes on the wire	**Description**
14 00 4A 67	PidTagMid (674A0014 [Int64])
01 00 00 00-00 78 48 C1	fixedSizeValue [Int64] -4519230284670959615
0B 00 02 00	propDef PidTagAlternateRecipientAllowed (0002000B [Bool])
01 00	fixedSizeValue [Bool] True
03 00 17 00	propDef PidTagImportance (00170003 [Int32])
01 00 00 00	fixedSizeValue [Int32] 1
1F 00 1A 00	propDef PidTagMessageClass (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 (0023000B [Bool])
00 00	fixedSizeValue [Bool] False
03 00 26 00	propDef PidTagPriority (00260003 [Int32])
00 00 00 00	fixedSizeValue [Int32] 0
0B 00 29 00	propDef PidTagReadReceiptRequested (0029000B [Bool])
00 00	fixedSizeValue [Bool] False
03 00 36 00	propDef PidTagSensitivity (00360003 [Int32])
00 00 00 00	fixedSizeValue [Int32] 0
1F 00 37 00	propDef PidTagSubject (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...
40 00 39 00	propDef PidTagClientSubmitTime (00390040 [SysTime])
00 B4 A1 9D-8B 84 C8 01	fixedSizeValue [SysTime] 2008-03-12T21:54:16.0000000
02 01 3B 00	propDef PidTagSentRepresentingSearchKey (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 (003D001F [Unicode])
02 00 00 00	length 2 (0x2)
00 00	varSizeValue ..

Bytes on the wire	**Description**
02 01 3F 00	propDef PidTagReceivedByEntryId (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 (0040001F [Unicode])
06 00 00 00	length 6 (0x6)
74 00 31 00-00 00	varSizeValue t.1...
02 01 41 00	propDef PidTagSentRepresentingEntryId (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 (0042001F [Unicode])
06 00 00 00	length 6 (0x6)
74 00 31 00-00 00	varSizeValue t.1...
02 01 43 00	propDef PidTagReceivedRepresentingEntryId (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 (0044001F [Unicode])
06 00 00 00	length 6 (0x6)
74 00 31 00-00 00	varSizeValue t.1...
02 01 51 00	propDef PidTagReceivedBySearchKey (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 ...	
	propDef

Bytes on the wire	**Description**
02 01 52 00	PidTagReceivedRepresentingSearchKey (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 (0063000B [Bool])
01 00	fixedSizeValue [Bool] True
1F 00 64 00	propDef PidTagSentRepresentingAddressType (0064001F [Unicode])
06 00 00 00	length 6 (0x6)
45 00 58 00-00 00	varSizeValue E.X...
1F 00 65 00	propDef PidTagSentRepresentingEmailAddress (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 (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 (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 (0075001F [Unicode])
06 00 00 00	length 6 (0x6)
45 00 58 00-00 00	varSizeValue E.X...
1F 00 76 00	propDef PidTagReceivedByEmailAddress (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 ...	

Bytes on the wire	**Description**
1F 00 77 00	propDef PidTagReceivedRepresentingAddressType (0077001F [Unicode])
06 00 00 00	length 6 (0x6)
45 00 58 00-00 00	varSizeValue E.X...
1F 00 78 00	propDef PidTagReceivedRepresentingEmailAddress (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 (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 00-66 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 ...	
0B 00 17 0C	propDef PidTagReplyRequested (0C17000B [Bool])
01 00	fixedSizeValue [Bool] True
02 01 19 0C	propDef PidTagSenderEntryId (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 (0C1A001F [Unicode])
06 00 00 00	length 6 (0x6)
74 00 31 00-00 00	varSizeValue t.1...
02 01 1D 0C	propDef PidTagSenderSearchKey (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 (0C1E001F [Unicode])

Bytes on the wire	**Description**
06 00 00 00	length 6 (0x6)
45 00 58 00-00 00	varSizeValue E.X...
1F 00 1F 0C	propDef PidTagSenderEmailAddress (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 (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 (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 (0x812C [PSETID_Task]) [Bool]
00 00	fixedSizeValue [Bool] False
40 00 06 0E	propDef PidTagMessageDeliveryTime (0E060040 [SysTime])
00 0E 04 A0-8B 84 C8 01	fixedSizeValue [SysTime] 2008-03-12T21:54:20.0000000
03 00 07 0E	propDef PidTagMessageFlags (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 (0x8110 [PSETID_Task]) [Int32]
00 00 00 00	fixedSizeValue [Int32] 0
03 00 17 0E	propDef PidTagMessageStatus (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 (0x8124 [PSETID_Task]) [Bool]
	fixedSizeValue

Bytes on the wire	**Description**
00 00	[Bool] False
1F 00 1D 0E	propDef PidTagNormalizedSubject (0E1D001F [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 1F 0E	propDef PidTagRtfInSync (0E1F000B [Bool])
01 00	fixedSizeValue [Bool] True
03 00 23 0E	propDef PidTagInternetArticleNumber (0E230003 [Int32])
1B 00 00 00	fixedSizeValue [Int32] 27
03 00 2B 0E	propDef PidTagToDoItemFlags (0E2B0003 [Int32])
01 00 00 00	fixedSizeValue [Int32] 1
03 00 79 0E	propDef PidTagTrustSender (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 (0x8111 [PSETID_Task]) [Int32]
00 00 00 00	fixedSizeValue [Int32] 0
03 00 F7 0F	propDef PidTagAccessLevel (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 (0x8126 [PSETID_Task]) [Bool]
00 00	fixedSizeValue [Bool] False
02 01 09 10	propDef PidTagRtfCompressed (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 (0x8103 [PSETID_Task]) [Bool]
00 00	fixedSizeValue [Bool] False
1F 00 35 10	propDef PidTagInternetMessageId (1035001F [Unicode])
	length

Bytes on the wire	**Description**
AC 00 00 00	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 (10800003 [Int32])
FF FF FF FF	fixedSizeValue [Int32] -1
03 00 90 10	propDef PidTagFlagStatus (10900003 [Int32])
02 00 00 00	fixedSizeValue [Int32] 2
03 00 95 10	propDef PidTagFollowupIcon (10950003 [Int32])
06 00 00 00	fixedSizeValue [Int32] 6
40 00 07 30	propDef PidTagCreationTime (30070040 [SysTime])
90 F8 65 B0-BC 84 C8 01	fixedSizeValue [SysTime] 2008-03-13T03:45:32.8250000
40 00 08 30	propDef PidTagLastModificationTime (30080040 [SysTime])
90 F8 65 B0-BC 84 C8 01	fixedSizeValue [SysTime] 2008-03-13T03:45:32.8250000
02 01 0B 30	propDef PidTagSearchKey (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 (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 PidTagSendRichInfo (3A40000B [Bool])
01 00	fixedSizeValue [Bool] True
03 00 DE 3F	propDef PidTagInternetCodepage (3FDE0003 [Int32])
9F 4E 00 00	fixedSizeValue [Int32] 20127
03 00 F1 3F	propDef PidTagMessageLocaleId (3FF10003 [Int32])
09 04 00 00	fixedSizeValue [Int32] 1033
1F 00 F8 3F	propDef PidTagCreatorName (3FF8001F [Unicode])
06 00 00 00	length 6 (0x6)
	varSizeValue

Bytes on the wire	**Description**
74 00 31 00-00 00	t.1...
1F 00 FA 3F	propDef PidTagLastModifierName (3FFA001F [Unicode])
06 00 00 00	length 6 (0x6)
74 00 31 00-00 00	varSizeValue t.1...
03 00 FD 3F	propDef PidTagMessageCodepage (3FFD0003 [Int32])
E3 04 00 00	fixedSizeValue [Int32] 1251
03 00 19 40	propDef PidTagSenderFlags (40190003 [Int32])
00 00 00 00	fixedSizeValue [Int32] 0
03 00 1A 40	propDef PidTagSentRepresentingFlags (401A0003 [Int32])
00 00 00 00	fixedSizeValue [Int32] 0
03 00 1B 40	propDef PidTagReceivedByFlags (401B0003 [Int32])
00 00 00 00	fixedSizeValue [Int32] 0
03 00 1C 40	propDef PidTagReceivedRepresentingFlags (401C0003 [Int32])
00 00 00 00	fixedSizeValue [Int32] 0
03 00 76 40	propDef PidTagContentFilterSpamConfidenceLevel (40760003 [Int32])
FF FF FF FF	fixedSizeValue [Int32] -1
03 00 02 59	propDef PidTagInternetMailOverrideFormat (59020003 [Int32])
00 00 16 00	fixedSizeValue [Int32] 1441792
03 00 09 59	propDef PidTagMessageEditorFormat (59090003 [Int32])
02 00 00 00	fixedSizeValue [Int32] 2
0B 00 4A 66	propDef PidTagHasNamedProperties (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 (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 (0x8503 [PSETID_Common]) [Bool]
00 00	fixedSizeValue [Bool] False
1F 00 1A 80-08 20 06 00 00 00 00 00-C0 00 00 00 00 00 00 46-00 A4 85 00 00	propDef PidLidToDoTitle (0x85A4 [PSETID_Common]) [Unicode]
	length

Bytes on the wire	**Description**
0E 00 00 00	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 (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 (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 (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 (0x8506 [PSETID_Common]) [Bool]
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 (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 (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 (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	propDef PidLidReminderDelta (0x8501 [PSETID_Common]) [Int32]

Bytes on the wire	**Description**
00	
00 00 00 00	fixedSizeValue [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 (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 (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 (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 (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 (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 (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 (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 (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 (0x8112 [PSETID_Task]) [Int32]

Bytes on the wire	**Description**
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 (0x8123 [PSETID_Task]) [Int32]
FF FF FF 7F	fixedSizeValue [Int32] 2147483647
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 (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 (0x8129 [PSETID_Task]) [Int32]
00 00 00 00	fixedSizeValue [Int32] 0
03 00 03 40	**marker** **StartRecip (40030003 [Int32])**
03 00 00 30	propDef PidTagRowid (30000003 [Int32])
00 00 00 00	fixedSizeValue [Int32] 0
1F 00 02 30	propDef PidTagAddressType (3002001F [Unicode])
06 00 00 00	length 6 (0x6)
45 00 58 00-00 00	varSizeValue E.X...
1F 00 03 30	propDef PidTagEmailAddress (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 00-49 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 01 30	propDef PidTagDisplayName (3001001F [Unicode])
06 00 00 00	length 6 (0x6)
74 00 31 00-00 00	varSizeValue t.1...
02 01 F6 0F	propDef PidTagInstanceKey (0FF60102 [Binary])
04 00 00 00	length 4 (0x4)
00 00 00 00	varSizeValue ....
03 00 15 0C	propDef PidTagRecipientType (0C150003 [Int32])
	fixedSizeValue

Bytes on the wire	**Description**
01 00 00 00	[Int32] 1
02 01 FF 0F	propDef PidTagEntryId (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 (300B0102 [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 20 3A	propDef PidTagTransmittableDisplayName (3A20001F [Unicode])
06 00 00 00	length 6 (0x6)
74 00 31 00-00 00	varSizeValue t.1...
0B 00 0F 0E	propDef PidTagResponsibility (0E0F000B [Bool])
01 00	fixedSizeValue [Bool] True
0B 00 40 3A	propDef PidTagSendRichInfo (3A40000B [Bool])
01 00	fixedSizeValue [Bool] True
03 00 FD 5F	propDef PidTagRecipientFlags (5FFD0003 [Int32])
01 00 00 00	fixedSizeValue [Int32] 1
02 01 F7 5F	propDef PidTagRecipientEntryId (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 ...	
1F 00 FE 39	propDef PidTagPrimarySmtpAddress (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 ...	

Bytes on the wire	**Description**
03 00 05 39	propDef PidTagDisplayTypeEx (39050003 [Int32])
00 00 00 40	fixedSizeValue [Int32] 1073741824
03 00 00 39	propDef PidTagDisplayType (39000003 [Int32])
00 00 00 00	fixedSizeValue [Int32] 0
03 00 FE 0F	propDef PidTagObjectType (0FFE0003 [Int32])
06 00 00 00	fixedSizeValue [Int32] 6
1F 00 FF 39	propDef PidTag7BitDisplayName (39FF001F [Unicode])
06 00 00 00	length 6 (0x6)
74 00 31 00-00 00	varSizeValue t.1...
1F 00 00 3A	propDef PidTagAccount (3A00001F [Unicode])
06 00 00 00	length 6 (0x6)
74 00 31 00-00 00	varSizeValue t.1...
03 00 DE 5F	propDef PidTagRecipientResourceState (5FDE0003 [Int32])
00 00 00 00	fixedSizeValue [Int32] 0
03 00 DF 5F	propDef PidTagRecipientOrder (5FDF0003 [Int32])
00 00 00 00	fixedSizeValue [Int32] 0
1F 00 F6 5F	propDef PidTagRecipientDisplayName (5FF6001F [Unicode])
06 00 00 00	length 6 (0x6)
74 00 31 00-00 00	varSizeValue t.1...
03 00 FF 5F	propDef PidTagRecipientTrackStatus (5FFF0003 [Int32])
00 00 00 00	fixedSizeValue [Int32] 0
03 00 04 40	**marker** **EndToRecip (40040003 [Int32])**
03 00 02 40	**marker** **EndEmbed (40020003 [Int32])**
03 00 0E 40	**marker** **EndAttach (400E0003 [Int32])**
03 00 13 40	**marker** **IncrSyncDel (40130003 [Int32])**
02 01 E5 67	propDef PidTagIdsetDeleted (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 (402F0003 [Int32])**
02 01 2D 40	propDef PidTagIdsetRead (402D0102 [Binary])
0A 00 00 00	length 10 (0xA)

Bytes on the wire	**Description**
01 00 06 00-00 00 78 2E 1F 00	varSizeValue .....x. ..
02 01 2E 40	propDef PidTagIdsetUnread (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 (403A0003 [Int32])**
02 01 96 67	propDef PidTagCnsetSeen (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 PidTagCnsetSeenFAI (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 PidTagIdsetGiven (40170003 [Int32])
38 00 00 00	length 56 (0x38)
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 80-01 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 PidTagCnsetRead (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 (403B0003 [Int32])**
03 00 14 40	**marker** **IncrSyncEnd (40140003 [Int32])**
	EOS

5 Security

5.1 Security Considerations for Implementers

Individual security considerations are specified in sections 3.2.4.5.2.6 and 3.2.4.5.2.7.

There are no additional security considerations specific to the Bulk Data Transfer protocol. Security considerations pertaining to the underlying Wire Format protocol, as specified in [MS-OXCRPC] section 5, do apply to this specification.

5.2 Index of Security Parameters

None.

6 Appendix A: Office/Exchange Behavior

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

- Microsoft Office 2003
- Microsoft Exchange 2003
- Microsoft Office 2007
- Microsoft Exchange 2007

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

<1> Section 2.2.1.1: Exchange Server 2003 SP2 and Exchange Server 2007 SP1 operate on the assumption that the **ICS state** properties are zero-length byte arrays if a client fails to send them when setting up a contents synchronization download. 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.

<2> Section 2.2.1.1.1: Clients MUST send this property with a property tag that defines it as **PtypInteger32**. Servers MAY accept this property even if the type is not specified as **PtypInteger32**.

<3> Section 2.2.1.1.1: **Uploading** this ICS State property into the **synchronization upload context** has no effect on the Exchange Server implementation of the protocol.

<4> Multiple: Exchange 2003 SP2 and Outlook 2003 SP3 do not support partial item downloads. Exchange 2003 SP2 does not recognize the **SendOptions** flag **PartialItem** and Outlook 2003 SP3 does not pass it.

<5>Section 2.2.3.1.1.1.2: Exchange 2003 SP2 and Exchange 2007 SP1 define additional flags for this enumeration, which are only used in server-to-server communications. For that reason, the **ROP** does not fail if those flags are passed from clients.

<6> Section 2.2.3.1.1.5: Outlook 2003 SP3 does not recognize this error code.

<7> Section 2.2.3.1.2.2: Clients MUST ignore the value of this field when communicating with Exchange 2003 SP2 or Exchange 2007 SP1.

<8> Section 2.2.3.1.2.2: Exchange 2003 SP2 and Exchange 2007 SP1 always set this field to 0x0000.

<9> Section 2.2.3.2.1.1.2: Exchange 2003 SP2 and Exchange 2007 SP1 MAY output bodies of embedded **messages** in compressed RTF.

<10> Section 2.2.3.2.1.1.2: Exchange 2003 SP2 and Exchange 2007 SP1 define additional flags for this enumeration, which are only used in server-to-server communications. For that reason, the **ROP** will not fail if those flags are passed from clients.

<11> Section 2.2.3.2.4.4: Exchange 2003 SP2 and Exchange 2007 SP1 do not support this **ROP** on public folders.

<12> Section 2.2.4.1.3: Exchange 2007 SP1 and below MAY fail to add a null-terminator. This difference will not be preserved in the later versions of Exchange.

<13> Section 3.1.1.2: In Exchange 2003 SP2 and Exchange 2007 SP1, property group mappings do not change frequently, but they do change with each version of Exchange Server. When a **message** is modified and the default mapping has changed after an upgrade, the property group mapping of the message is updated.

<14> Section 3.1.3.2.2: Exchange 2003 SP2, Exchange 2007 SP1, Outlook 2003 SP3, and Outlook 2007 SP1 only perform this step for **messages**. For **folders**, Exchange 2003 SP2 and Exchange 2007 SP1 keep a server version if the client version is in conflict, but has the same value of for the **PidTagLastModificationTime** property.

<15> Section 3.3.4.2.2: Outlook 2003 SP3 and Outlook 2007 SP1 upload initial **ICS state** and **download** the final/checkpoint ICS state when doing synchronization uploads.

<16> Section 3.3.4.2.2.5: Online mode **ROPs** are used by Outlook 2003 SP3 and Outlook 2007 SP1 when they are in online mode and are connected to the server. These ROPs are specified in [MS-OXCFOLD] and [MS-OXCMSG].

Index