[MS-OXCNOTIF]:
Core Notifications Protocol

Intellectual Property Rights Notice for Open Specifications Documentation

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

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

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

Revision Summary

Date	Revision History	Revision Class	Comments
04/04/2008	0.1		Initial Availability.
06/27/2008	1.0		Initial Release.
08/06/2008	1.01		Revised and edited technical content.
09/03/2008	1.02		Revised and edited technical content.
12/03/2008	1.03		Minor editorial fixes.
03/04/2009	1.04		Revised and edited technical content.
04/10/2009	2.0		Updated technical content and applicable product releases.
07/15/2009	3.0	Major	Revised and edited for technical content.
11/04/2009	3.1.0	Minor	Updated the technical content.
02/10/2010	4.0.0	Major	Updated and revised the technical content.
05/05/2010	4.1.0	Minor	Updated the technical content.
08/04/2010	5.0	Major	Significantly changed the technical content.
11/03/2010	5.0	No change	No changes to the meaning, language, or formatting of the technical content.
03/18/2011	6.0	Major	Significantly changed the technical content.
08/05/2011	6.0	No change	No changes to the meaning, language, or formatting of the technical content.
10/07/2011	7.0	Major	Significantly changed the technical content.
01/20/2012	8.0	Major	Significantly changed the technical content.
04/27/2012	9.0	Major	Significantly changed the technical content.
07/16/2012	10.0	Major	Significantly changed the technical content.
10/08/2012	10.1	Minor	Clarified the meaning of the technical content.
02/11/2013	10.1	No change	No changes to the meaning, language, or formatting of the technical content.
07/26/2013	10.1	No change	No changes to the meaning, language, or formatting of the technical content.
11/18/2013	11.0	Major	Significantly changed the technical content.
02/10/2014	11.0	No change	No changes to the meaning, language, or formatting of the technical content.

Date	Revision History	Revision Class	Comments
04/30/2014	12.0	Major	Significantly changed the technical content.
07/31/2014	12.0	No change	No changes to the meaning, language, or formatting of the technical content.

Table of Contents

1 Introduction

The Core Notifications Protocol transmits notifications to a client about specific events on a server. This protocol is commonly used to inform the client about changes that have occurred in folders and messages on the server.

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

1.1 Glossary

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

ASCII
datagram
handle
Hypertext Transfer Protocol (HTTP)
Internet Protocol version 6 (IPv6)
remote procedure call (RPC)
Unicode
User Datagram Protocol (UDP)

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

asynchronous context handle
binary large object (BLOB)
mailbox
message class
Messaging Application Programming Interface (MAPI)
remote operation (ROP)
ROP request
ROP request buffer
ROP response
ROP response buffer
search folder
session context handle
Table object

The following terms are specific to this document:

callback address: An object that encapsulates an Internet address that is registered by a client and that a server can use for push notifications.

notification subscription: A request to receive notifications from a server when specific events occur on that server.

outstanding RPC call: An asynchronous remote procedure call (RPC) that has not been completed by a server yet.

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

References to Microsoft Open Specification documents do not include a publishing year because links are to the latest version of the documents, which are updated frequently. References to other documents include a publishing year when one is available.

1.2.1 Normative References

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

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

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

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

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

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

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

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

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

[RFC768] Postel, J., "User Datagram Protocol", STD 6, RFC 768, August 1980, http://www.ietf.org/rfc/rfc768.txt

1.2.2 Informative References

[MSDN-ENM] Microsoft Corporation, "Event Notification in MAPI", http://msdn.microsoft.com/en-us/library/ms528269(EXCHG.10).aspx

[MSDN-WS2] Microsoft Corporation, "Windows Sockets 2", http://msdn.microsoft.com/en-us/library/ms740673(VS.85).aspx

[MSFT-ConfigStaticUDPPort] Microsoft Corporation, "Configure a Static UDP Port for Push Notifications in an Exchange 2010 Environment (en-US)", http://social.technet.microsoft.com/wiki/contents/articles/2542.configure-a-static-udp-port-for-push-notifications-in-an-exchange-2010-environment.aspx

[MS-GLOS] Microsoft Corporation, "Windows Protocols Master Glossary".

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

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

1.3 Overview

This protocol enables a client to receive notifications about specific events that occur on the messaging server. The client can subscribe to certain events on the server, and when one of the events occurs, the server sends the client a notification. The notification sent by the server is

commonly a two part operation. First, the server notifies the client about pending notifications. Then the server transmits the notification details.

The server supports three methods for notifying the client of pending notifications on the server:

- Asynchronous **RPC** notifications. This method enables the client to make an asynchronous remote procedure call (RPC) call to the server; the server does not complete the RPC call until there is a notification for the session.
- Asynchronous notifications via **HTTP** extensions, as described in [MS-OXCMAPIHTTP].
- Push notifications. This method relies on a **callback address** being registered with the server, so that **User Datagram Protocol (UDP) datagrams** can be sent to the callback address when pending notifications exist.
- The **RopPending ROP** ([MS-OXCROPS] section 2.2.14.3). This **ROP** is included in the **EcDoRpcExt2** method call response if there are pending notifications on the server and the details of the notification do not fit in the response buffer.

Regardless of the means used to notify the client of the pending notification, the notification details are sent to the client by using the **RopNotify** ROP (section 2.2.1.4.1).

1.4 Relationship to Other Protocols

This specification provides a low-level explanation of notifying a client about events on the server. For information about applying this protocol in a **Messaging Application Programming Interface (MAPI)** provider, see [MSDN-ENM].

This specification relies on an understanding of [MS-OXCRPC] , [MS-OXCMAPIHTTP], and [MS-OXCROPS].

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

1.5 Prerequisites/Preconditions

This specification assumes that the client has previously logged on to the server and created a session context.

1.6 Applicability Statement

This protocol was designed to be used for the following purposes:

- Notifying clients about specific events on the server.
- Notifying clients about notifications pending for the client on the server.

This protocol provides basic information, a high degree of efficiency, and complete preservation of data fidelity for these uses. Note, however, that it might not be appropriate for use in scenarios that do any of the following:

- Require replication of mailbox content between clients and servers.
- Require client-driven copying of data between different mailboxes on different servers.
- Require exporting or importing of data to or from a mailbox.

1.7 Versioning and Capability Negotiation

This specification covers versioning issues in the following areas:

- Supported Transports: This protocol uses the Wire Format Protocol, as described in [MS-OXCRPC], the Remote Operations (ROP) List and Encoding Protocol, as described in [MS-OXCROPS], the MAPI extensions to HTTP, as described in [MS-OXCMAPIHTTP], and Internet protocols as described in section 2.1.
- Protocol Versions: This protocol has only one interface version.
- Capability Negotiation: The protocol does not require asynchronous RPC notifications to be implemented. The client examines the server version to determine whether asynchronous RPC notifications are supported. For more information about how to determine server version, see [MS-OXCRPC].
- Localization: This protocol passes text strings in notification details. Localization considerations for such strings are described in [MS-OXCMSG] section 2.2.1.3.

1.8 Vendor-Extensible Fields

None.

1.9 Standards Assignments

None.

2 Messages

2.1 Transport

The commands specified by this protocol are sent to and received from the server by using the underlying **ROP request buffers** and **ROP response buffers**, respectively, as specified in [MS-OXCROPS].

Asynchronous calls are made on the server by using RPC transport, as specified in [MS-OXCRPC], and the MAPI extensions to HTTP<1>, as specified in [MS-OXCMAPIHTTP].

UDP datagrams are sent from server to client by using the User Datagram Protocol (UDP), as specified in [RFC768]. For more information, see [MSDN-WS2].

2.2 Message Syntax

2.2.1 Notifications

This section specifies the following:

- The server events that the client can be notified of.
- The **RopRegisterNotification** ROP, which is used to subscribe to notifications.
- The ROPs and RPCs used to notify the client of pending notifications:
  - The **EcDoAsyncConnectEx** method.
  - The **EcDoAsyncWaitEx** method.
  - The **EcRRegisterPushNotification** method.
  - The **RopPending** ROP ([MS-OXCROPS] section 2.2.14.3).
  - The **NotificationWait** request type<2> ([MS-OXCMAPIHTTP] section 2.2.4.4).
- The **RopNotify** ROP (section 2.2.1.4.1), which is used to send notification details.

2.2.1.1 Server Event Types

The following table specifies the events that happen on the server. Clients can register to receive notifications about these events by using the **RopRegisterNotification** ROP (section 2.2.1.2.1).

Event name	Description
NewMail	A new email message has been received by the server.
ObjectCopied	An existing object has been copied on the server.
ObjectCreated	A new object has been created on the server.
ObjectDeleted	An existing object has been deleted from the server.
ObjectModified	An existing object has been modified on the server.
ObjectMoved	An existing object has been moved to another location on the server.

Event name	Description
SearchComplete	A search operation has been completed on the server.
TableModified	A table has been modified on the server. **TableModified** event types are specified in section 2.2.1.1.1.

2.2.1.1.1 TableModified Event Types

The following table specifies the table modification event types. Clients can register to receive notifications about these events by using the **RopRegisterNotification** ROP (section 2.2.1.2.1).

Event name	Description
TableChanged	A table has been changed.
TableRowAdded	A new row has been added to the table.
TableRowDeleted	An existing row has been deleted from the table.
TableRowModified	An existing row has been modified in the table.
TableRestrictionChanged	A table restriction has been cleared, removed, or is pending. For more details about how this event type is related to the TBLSTAT_RESTRICTING value of the **TableStatus** field, as specified in [MS-OXCTABL] section 2.2.2.1.3, of the **RopRestrict** ROP ([MS-OXCROPS] section 2.2.5.3), see [MS-OXCTABL] section 3.2.5.1.

For server steps related to creating and sending **TableModified** event notifications, see section 3.1.4.3. For client initialization steps related to subscribing to **TableModified** event notifications, see section 3.2.4.2.

2.2.1.2 Subscription Management

Subscription management is handled by the **RopRegisterNotification** ROP (section 2.2.1.2.1). For more information about how clients subscribe to notification events, see section 3.2.4.1.

2.2.1.2.1 RopRegisterNotification ROP

The **RopRegisterNotification** ROP ([MS-OXCROPS] section 2.2.14.1) creates a subscription for specified notifications on the server and returns a **handle** of the subscription to the client.

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

2.2.1.2.1.1 RopRegisterNotification ROP Request Buffer

The following descriptions define valid fields for the request buffer of the **RopRegisterNotification** ROP ([MS-OXCROPS] section 2.2.14.1).

NotificationTypes (1 byte): A set of bits describing notifications that the client is interested in receiving.

The following table lists the values that are available for notification types.

Value	Meaning
0x02	The server sends notifications to the client when **NewMail** events occur within the scope of interest.
0x04	The server sends notifications to the client when **ObjectCreated** events occur within the scope of interest.
0x08	The server sends notifications to the client when **ObjectDeleted** events occur within the scope of interest.
0x10	The server sends notifications to the client when **ObjectModified** events occur within the scope of interest.
0x20	The server sends notifications to the client when **ObjectMoved** events occur within the scope of interest.
0x40	The server sends notifications to the client when **ObjectCopied** events occur within the scope of interest.
0x80	The server sends notifications to the client when **SearchComplete** events occur within the scope of interest.

For details about server events, see section 2.2.1.1.

Reserved (1 byte): This field is reserved. The field value MUST be 0x00. The server behavior is undefined if the value is not 0x00.

WantWholeStore (1 byte): A value of **TRUE** (0x01) if the scope for notifications is the entire **mailbox**; otherwise, **FALSE** (0x00).

2.2.1.3 Pending Notifications

Pending notifications rely on transmission of one or more of the following methods:

- The **EcDoAsyncConnectEx** method, which is used in asynchronous RPC notifications.
- The **EcDoAsyncWaitEx** method, which is also used in asynchronous RPC notifications.
- The **EcRRegisterPushNotification** method, which is used for push notifications.
- The **EcDoRpcExt2** method and the **RopPending** ROP ([MS-OXCROPS] section 2.2.14.3).

2.2.1.3.1 EcDoAsyncConnectEx Method

The **EcDoAsyncConnectEx**<3> RPC method, as specified in [MS-OXCRPC] section 3.1.4.4, is used to acquire an **asynchronous context handle** on the server to use in subsequent **EcDoAsyncWaitEx** method calls, as specified in [MS-OXCRPC] section 3.3.4.1. The **EcDoAsyncConnectEx** method is used to support asynchronous RPC notifications. For more information about how the client sends **EcDoAsyncConnectEx** method to initialize the notification process, see section 3.2.5.2. For more information about how the server receives the **EcDoAsyncConnectEx** method, see section 3.1.5.2.

2.2.1.3.2 EcDoAsyncWaitEx Method

The **EcDoAsyncWaitEx**<4> asynchronous RPC method, as specified in [MS-OXCRPC] section 3.3.4.1, is used to inform a client about pending notifications on the server. The **EcDoAsyncWaitEx**

method is used to support asynchronous RPC notifications. For more information about how the client sends and receives **EcDoAsyncWaitEx** method calls, see section 3.2.5.5.1. For more information about how the server receives and completes **EcDoAsyncWaitEx** method calls, see section 3.1.5.3.

2.2.1.3.3 EcRRegisterPushNotification Method

The **EcRRegisterPushNotification**<5> RPC method, as specified in [MS-OXCRPC] section 3.1.4.5, is used to register a callback address of a client on the server. The callback address is required in order to receive UDP datagrams from the server, and is used to support push notifications, which is one way in which the server can notify clients of pending notifications. The UDP datagrams inform the client that notifications are pending on the server for the session.

2.2.1.3.4 RopPending ROP

The **RopPending** ROP ([MS-OXCROPS] section 2.2.14.3) notifies the client that there are pending notifications on the server for the client. This ROP MUST appear only in response buffers of either the **EcDoRpcExt2** method, as specified in [MS-OXCRPC] section 3.1.4.2, or the **Execute** request type,<6> as specified in [MS-OXCMAPIHTTP] section 2.2.4.2.2. For more information about how the server sends this **ROP response**, see section 3.1.5.6. For more information about how the client receives this ROP response, see section 3.2.5.5.3.

2.2.1.4 Notification Details

Notification details are transmitted by using the **RopNotify** ROP (section 2.2.1.4.1).

2.2.1.4.1 RopNotify ROP

The **RopNotify** ROP ([MS-OXCROPS] section 2.2.14.2) provides the client with the details of notifications that are sent by server. This ROP MUST appear only in response buffers of the **EcDoRpcExt2** method, as specified in [MS-OXCRPC] section 3.1.4.2, or in the **Execute** request type success response body<7>, as specified in [MS-OXCMAPIHTTP] section 2.2.4.2.2.

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

For more information about how the server sends notification details using the **RopNotify** ROP, see section 3.1.5.7. For more information about how the client receives notification details using the **RopNotify** ROP, see section 3.2.5.7.

2.2.1.4.1.1 RopNotify ROP Response Buffer

The following descriptions define valid fields for the response buffer of the **RopNotify** ROP ([MS-OXCROPS] section 2.2.14.2).

NotificationHandle (4 bytes): The 32-bit server object handle of the target object for the notification. The target object can be a **notification subscription** or a table.

NotificationData (variable): A structure containing the following fields.

NotificationFlags (2 bytes): A combination of an enumeration and flags that describe the type of the notification and the availability of the notification data fields.

The layout is shown in the following table.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
NotificationType												T	U	S	M																

NotificationType (12 bits): NotificationType is a 12-bit enumeration defining the type of the notification. The possible values for this enumeration are listed in the following table.

Value	Meaning
0x0002	The notification is for a **NewMail** event.
0x0004	The notification is for an **ObjectCreated** event.
0x0008	The notification is for an **ObjectDeleted** event.
0x0010	The notification is for an **ObjectModified** event.
0x0020	The notification is for an **ObjectMoved** event.
0x0040	The notification is for an **ObjectCopied** event.
0x0080	The notification is for a **SearchCompleted** event.
0x0100	The notification is for a **TableModified** event.
0x0400	This value is reserved.

T (1 bit): Bitmask = 0x1000. The notification contains information about a change in the total number of messages in a folder triggering the event. If this bit is set, **NotificationType** MUST be 0x0010.

U (1 bit): Bitmask = 0x2000. The notification contains information about a change in the number of unread messages in a folder triggering the event. If this bit is set, **NotificationType** MUST be 0x0010.

S (1 bit): Bitmask = 0x4000. The notification is caused by an event in a **search folder (1)**. If this bit is set, bit 0x8000 MUST be set.

M (1 bit): Bitmask = 0x8000. The notification is caused by an event on a message.

TableEventType (2 bytes): A subtype of the notification for a **TableModified** event. This field is available only if the **NotificationType** value in the **NotificationFlags** field is 0x0100.

The following table lists the values that are available for event types. For more details, see section 2.2.1.1.1.

Value	Meaning
0x0001	The notification is for **TableChanged** events.
0x0003	The notification is for **TableRowAdded** events.
0x0004	The notification is for **TableRowDeleted** events.
0x0005	The notification is for **TableRowModified** events.

Value	Meaning
0x0007	The notification is for **TableRestrictionChanged** events.

TableRowFolderID (8 bytes): The value of the **Folder ID** structure, as specified in [MS-OXCDATA] section 2.2.1.1, of the item triggering the notification. This field is available only if the **TableEventType** field is available and is 0x0003, 0x0004, or 0x0005.

TableRowMessageID (8 bytes): The value of the **Message ID** structure, as specified in [MS-OXCDATA] section 2.2.1.2, of the item triggering the notification. This field is available only if bit 0x8000 is set in the **NotificationFlags** field and if the **TableEventType** field is available and is 0x0003, 0x0004, or 0x0005.

TableRowInstance (4 bytes): An identifier of the instance of the previous row in the table. This field is available only if bit 0x8000 is set in the **NotificationFlags** field and if the **TableEventType** field is available and is 0x0003, 0x0004, or 0x0005.

InsertAfterTableRowFolderID (8 bytes): The old value of the **Folder ID** structure of the item triggering the notification. This field is available only if the **TableEventType** field is available and is 0x0003 or 0x0005.

InsertAfterTableRowID (8 bytes): The old value of the **Message ID** structure of the item triggering the notification. This field is available only if bit 0x8000 is set in the **NotificationFlags** field and if the **TableEventType** field is available and is 0x0003 or 0x0005.

InsertAfterTableRowInstance (4 bytes): An unsigned 32-bit identifier of the instance of the row where the modified row is inserted.

TableRowDataSize (2 bytes): An unsigned 16-bit integer that indicates the length of the table row data. This field is available only if the **TableEventType** field is available and is 0x0003 or 0x0005.

TableRowData (variable): The table row data, which contains a list of property values for the row that was added or modified in the table. This field is available only if the **TableEventType** field is available and is 0x0003 or 0x0005.

FolderId (8 bytes): The **Folder ID** structure of the item triggering the event. This field is available only if the **NotificationType** value in the **NotificationFlags** field is not 0x0100 or 0x0400.

MessageId (8 bytes): The **Message ID** structure, as specified in [MS-OXCDATA] section 2.2.1.2, of the item triggering the event. This field is available only if the **NotificationType** value in the **NotificationFlags** field is not 0x0100 or 0x0400, and bit 0x8000 is set in the **NotificationFlags** field.

ParentFolderId (8 bytes): The **Folder ID** structure of the parent folder of the item triggering the event. This field is available only if the value of the **NotificationType** field is 0x0004, 0x0008, 0x0020, or 0x0040, and it is sent for either a message in a search folder (1) (both bit 0x4000 and bit 0x8000 are set in the **NotificationFlags** field) or a folder (both bit 0x4000 and bit 0x8000 are not set in the **NotificationFlags** field).

OldFolderId (8 bytes): The old **Folder ID** structure of the item triggering the event. This field is available only if the **NotificationType** value in the **NotificationFlags** field is 0x0020 or 0x0040.

OldMessageId (8 bytes): The old **Message ID** structure of the item triggering the event. This field is available only if the value of the **NotificationType** field in the **NotificationFlags** field is 0x0020 or 0x0040 and bit 0x8000 is set in the **NotificationFlags** field.

OldParentFolderId (8 bytes): The old parent **Folder ID** structure of the item triggering the event. This field is available only if the value of the **NotificationType** field in the **NotificationFlags** field is 0x0020 or 0x0040 and bit 0x8000 is not set in the **NotificationFlags** field.

TagCount (2 bytes): An unsigned 16-bit integer that specifies the number of property tags in the **Tags** field. This field is available only if the value of the **NotificationType** field in the **NotificationFlags** field is 0x0004 or 0x0010. If the value of the **NotificationType** field in the **NotificationFlags** field is 0x0010, the value of this field SHOULD<8> be set to 0x0000. A value of 0xFFFF is returned if the number of tags to fit in the response creates a response that exceeds the maximum size of the output buffer, as defined by the **pcbOut** field in the **EcDoRpcExt2** RPC, as specified in [MS-OXCRPC] section 3.1.4.2, or in the **Execute** request type request body<9>, as specified in [MS-OXCMAPIHTTP] section 2.2.4.2.1. When OxFFFF is returned, the list of property tags is omitted.

Tags (variable): An array of unsigned 32-bit integers that identifies the IDs of properties that have changed. This field is available only if the **TagCount** field is available and the value of the **TagCount** field is not 0x0000 or 0xFFFF.

TotalMessageCount (4 bytes): An unsigned 32-bit integer that specifies the total number of items in the folder triggering this event. This field is available only if bit 0x1000 is set in the **NotificationFlags** field.

UnreadMessageCount (4 bytes): An unsigned 32-bit integer that specifies the number of unread items in a folder triggering this event. This field is available only if bit 0x2000 is set in the **NotificationFlags** field.

MessageFlags (4 bytes): An unsigned 32-bit integer that specifies the message flags of new mail that has been received. This field is available only if the value of the **NotificationType** field in the **NotificationFlags** field is 0x0002. For details, see [MS-OXCMSG] section 2.2.1.6.

UnicodeFlag (1 byte): A value of **TRUE** (0x01) indicates the value of the **MessageClass** field is in **Unicode**; otherwise, **FALSE** (0x00) indicates the value of the **MessageClass** field is in **ASCII**. A value of **FALSE** is returned if the client is working in cached mode, as specified by the **ClientMode** field in [MS-OXCRPC] section 2.2.2.2.4. This field is available only if the value of the **NotificationType** field in the **NotificationFlags** field is 0x0002.

MessageClass (variable): A null-terminated string containing the **message class** of the new mail. The string is in Unicode if the **UnicodeFlag** field is set to **TRUE** (0x01). The string is in ASCII if **UnicodeFlag** is set to **FALSE** (0x00). This field is available only if the value of the **NotificationType** field in the **NotificationFlags** field is 0x0002.

3 Protocol Details

3.1 Server Details

3.1.1 Abstract Data Model

This section describes a conceptual model of possible data organization that an implementation maintains to participate in this protocol. The described organization is provided to facilitate the explanation of how the protocol behaves. This specification does not mandate that implementations adhere to this model as long as their external behavior is consistent with that described in this document.

This protocol includes the following abstract data model (ADM) elements:

Global.Handle, as specified in [MS-OXCRPC] section 3.1.1.

Global.AsynchronousHandle, as specified in [MS-OXCRPC] section 3.3.1.

session context cookie<10>**,** as specified in [MS-OXCMAPIHTTP] section 3.2.1.

The following ADM types are defined in this section:

NotificationSubscriptionObject: An object on the server associated with the session context that manages event notifications and notification subscriptions.

3.1.2 Timers

If push notifications are supported by the server, as specified in section 3.1.5.4, the server SHOULD allow for a 60-second interval between UDP datagrams until the client has retrieved all event information for the session. The server MUST provide server administrators a means to configure the time interval between the UDP datagrams.

3.1.3 Initialization

None.

3.1.4 Higher-Layer Triggered Events

3.1.4.1 Sending Pending Notifications

The server notifies the client of pending notifications in one of three ways: by completing an asynchronous **EcDoAsyncWaitEx** RPC method call, by using push notifications and sending a UDP datagram to a callback address, or by sending a **RopPending** ROP response ([MS-OXCROPS] section 2.2.14.3). For more details about responding to an asynchronous RPC call, see section 3.1.5.2. For more details about using push notifications and sending a UDP datagram, see section 3.1.5.4. For more details about sending a **RopPending** ROP response, see section 3.1.5.6.

3.1.4.2 Sending Notification Details

The server sends notification details to the client by sending the **RopNotify** ROP response (section 2.2.1.4.1). The **RopNotify** command is the only method to transmit notification details to the client, so it is used regardless of the method used to notify the client of the pending notification. For more details about sending the **RopNotify** ROP, see section 3.1.5.7.

3.1.4.3 Creating and Sending TableModified Event Notifications

If the client has subscribed to **TableModified** event notifications, by using the **RopRegisterNotification** ROP (section 2.2.1.2.1), the server SHOULD<11> require that a table view is created in order to send the **TableModified** event notifications, as specified in section 2.2.1.1.1. If a table view is required on the server, the server MUST receive a request from one of the following ROPs, each of which cause a table view to be created on the server: **RopCollapseRow** ([MS-OXCROPS] section 2.2.5.17), **RopExpandRow** ([MS-OXCROPS] section 2.2.5.16), **RopFindRow** ([MS-OXCROPS] section 2.2.5.13), **RopQueryColumnsAll** ([MS-OXCROPS] section 2.2.5.12), **RopQueryPosition** ([MS-OXCROPS] section 2.2.5.7), **RopQueryRows** ([MS-OXCROPS] section 2.2.5.4), **RopSeekRow** ([MS-OXCROPS] section 2.2.5.8), **RopSeekRowFractional** ([MS-OXCROPS] section 2.2.5.10), and **RopSeekRowBookmark** ([MS-OXCROPS] section 2.2.5.9). The server SHOULD then create a subscription to **TableModified** event notifications automatically for every table created on the server. The server MUST NOT create a subscription to table notifications for the tables that were created with a **NoNotifications** flag. For more details about the **NoNotifications** flag, see [MS-OXCFOLD] section 2.2.1.14.1 and [MS-OXCFOLD] section 2.2.1.13.1.

When a **TableModified** event occurs, the server generates a notification using one of the following three methods, listed in descending order of usefulness to the client.

1. The server generates an informative notification that specifies the nature of the change (content or hierarchy), the value of the **Folder ID** structure, as specified in [MS-OXCDATA] section 2.2.1.1, the value of the **Message ID** structure, as specified in [MS-OXCDATA] section 2.2.1.2, and new table values. The **TableRowAdded**, **TableRowDeleted**, and **TableRowModified** events each produce informative notifications.

2. The server generates a basic notification that does not include specifics about the change made. The **TableChanged** and **TableRestrictionChanged** events produce basic notifications.

3. The server does not generate a notification at all.

The notification level is server implementation-specific; however, the server SHOULD generate informative notifications whenever possible and only generate a basic notification when it is not feasible to generate an informative notification.

The server SHOULD<12> stop sending notifications if the **RopResetTable** ROP ([MS-OXCROPS] section 2.2.5.15) is received, until a new table view is created using one of the following ROPs: **RopCollapseRow**, **RopExpandRow**, **RopFindRow**, **RopQueryColumnsAll**, **RopQueryPosition**, **RopQueryRows**, **RopSeekRow**, **RopSeekRowFractional**, or **RopSeekRowBookmark**.

3.1.5 Message Processing Events and Sequencing Rules

3.1.5.1 Receiving a RopRegisterNotification ROP Request

When a **RopRegisterNotification** ROP (section 2.2.1.2.1) message is received by the server, the server SHOULD create a new Notification Subscription object and associate it with the session context. The server SHOULD save the information provided in the **RopRegisterNotification ROP request** fields for future use.

The server SHOULD allow multiple Notification Subscription objects to be created and associated with the same session context.

For details about how the client sends the **RopRegisterNotification** ROP request, see section 3.2.5.1.

3.1.5.2 Receiving an EcDoAsyncConnectEx Method Call

The server SHOULD<13> support the **EcDoAsyncConnectEx** method call, as specified in [MS-OXCRPC] section 3.1.4.4.

When a call to the **EcDoAsyncConnectEx** RPC, as specified in [MS-OXCRPC] section 3.1.4.4, is received by the server, the server MUST create an asynchronous context handle and MUST bind it to the **session context handle** used to make the call.

3.1.5.3 Receiving an EcDoAsyncWaitEx Method Call

The server SHOULD<14> support the **EcDoAsyncWaitEx** method call, as specified in [MS-OXCRPC] section 3.3.4.1.

Whenever an asynchronous **EcDoAsyncWaitEx** method call, as specified in [MS-OXCRPC] section 3.3.4.1, on the **AsyncEMSMDB** interface is received by the server, the server MUST validate that the asynchronous context handle provided is a valid asynchronous context handle that was returned from the **EcDoAsyncConnectEx** method call, as specified in [MS-OXCRPC] section 3.1.4.4. The server SHOULD NOT complete the call until there is a notification for the client session, or the call has been outstanding on the server 5 minutes. If the server already has a call outstanding for the same session context handle, the server SHOULD complete the new call and set the **ErrorCode** field to **Rejected**, as specified in [MS-OXCRPC] section 3.3.4.1, if another asynchronous RPC call is currently in progress on the server.

If the server completes the **outstanding RPC call** when there is a notification for the client session, the server MUST return the value **NotificationPending** in the **pulFlagsOut** field. The server MUST return 0x00000000 in the **pulFlagsOut** field if the call was completed when there is no notification for the client session.

3.1.5.4 Receiving an EcRRegisterPushNotification Method Call

The server MAY<15> support the **EcRRegisterPushNotification** method call, as specified in [MS-OXCRPC] section 3.1.4.5.

When a call to the **EcRRegisterPushNotification** method is received by the server, a valid callback address in the **rgbCallbackAddress** field and buffer with opaque client data in the **rgbContext** field MUST be present. The server MUST fail the call and MUST NOT take any actions if the callback address is not a valid **SOCKADDR** structure. For more information, see [MSDN-WS2].

The server SHOULD support at a minimum the AF_INET address type for IP support and the AF_INET6 address type for **IPv6** support.

The server MUST save the callback address and opaque context data on the session context for future use.

After the callback address has been successfully registered with the server, the server SHOULD send a UDP datagram containing the client's opaque data, from the **rgbContext** field, when a notification becomes available for the client.

If the server supports sending push notification UDP datagrams, the server MUST continue sending a UDP datagram to the callback address at 60-second intervals if event details are still queued for the client. The server SHOULD stop sending UDP datagrams only when all of the notifications have been retrieved from the server through **EcDoRpcExt2** method calls, as specified in [MS-OXCRPC] section 3.1.4.2.

3.1.5.5 Receiving an EcDoRpcExt2 Method Call

When the server receives an **EcDoRpcExt2** method call, as specified in [MS-OXCRPC] section 3.1.4.2, if there are pending notifications on the server, the server SHOULD send a **RopNotify** ROP response (section 2.2.1.4.1) for each pending notification on the session context that is associated with the client. If all the **RopNotify** ROP responses do not fit in the response buffer, the server SHOULD include as many **RopNotify** ROP responses as will fit in the response, and then include a **RopPending** ROP response ([MS-OXCROPS] section 2.2.14.3) to indicate that additional notifications are available on the server. For more details, see section 3.1.5.7 and section 3.1.5.7.

The server does not require that the **EcDoRpcExt2** method call include a ROP request.

3.1.5.6 Sending a RopPending ROP Response

The server SHOULD send a **RopPending** ROP response (section 2.2.1.3.4) to the client whenever there are pending notifications on the session context associated with the client and the **RopNotify** ROP response (section 2.2.1.4.1) for the associated notification does not fit in the response buffer. The server MAY<16> send a **RopPending** ROP response to the client whenever there are pending notifications on any linked session contexts.

3.1.5.7 Sending a RopNotify ROP Response

The server SHOULD send a **RopNotify** ROP response (section 2.2.1.4.1) to the client for each pending notification on the session context that is associated with the client. The server SHOULD send as many **RopNotify** ROP responses as the response buffer allows. If the server was not able to fit the details for all pending notifications into the response buffer using **RopNotify** ROP responses, it SHOULD include a **RopPending** ROP response (section 2.2.1.3.4) to indicate there are additional notifications available on the server, if the **RopPending** ROP response fits in the response buffer.

3.1.6 Timer Events

None.

3.1.7 Other Local Events

The server events and table events specified in section 2.2.1.1 and section 2.2.1.1.1 that occur on the server cause the pending notifications and notification detail messages to be sent. How the server triggers each of these events is implementation dependent and external to this protocol.

3.2 Client Details

3.2.1 Abstract Data Model

None.

3.2.2 Timers

None.

3.2.3 Initialization

Protocol initialization occurs when a client sends a request to the server to subscribe to notifications from the server, as specified in section 3.2.4.1.

3.2.4 Higher-Layer Triggered Events

The following sections specify the client-side higher-layer triggered events for this protocol.

3.2.4.1 Subscribing to Notifications

The client sends the **RopRegisterNotification** ROP request (section 2.2.1.2.1) to the server to subscribe to all notifications specified in section 2.2.1.1.

For more details about sending the **RopRegisterNotification** ROP request, see section 3.2.5.1.

3.2.4.2 Subscribing to TableModified Event Notifications

For a client to receive **TableModified** event notifications, in addition to sending the **RopRegisterNotification** ROP request, the client SHOULD<17> also send the one of the following ROPs to the server, which causes a table view to be created: **RopCollapseRow** ([MS-OXCROPS] section 2.2.5.17), **RopExpandRow** ([MS-OXCROPS] section 2.2.5.16), **RopFindRow** ([MS-OXCROPS] section 2.2.5.13), **RopQueryColumnsAll** ([MS-OXCROPS] section 2.2.5.12), **RopQueryPosition** ([MS-OXCROPS] section 2.2.5.7), **RopQueryRows** ([MS-OXCROPS] section 2.2.5.4), **RopSeekRow** ([MS-OXCROPS] section 2.2.5.8), **RopSeekRowFractional** ([MS-OXCROPS] section 2.2.5.10), and **RopSeekRowBookmark** ([MS-OXCROPS] section 2.2.5.9). Once a table view has been created, the client will receive **TableModified** event notifications so long as the **NoNotifications** flag has not been set on the table. The **NoNotifications** flag is specified in [MS-OXCFOLD] section 2.2.1.14.1 and [MS-OXCFOLD] section 2.2.1.13.1.

If the client sends the **RopResetTable** ROP ([MS-OXCROPS] section 2.2.5.15), the client SHOULD<18> stop receiving table notifications until one of the following ROPs is sent: **RopCollapseRow**, **RopExpandRow**, **RopFindRow**, **RopQueryColumnsAll**, **RopQueryPosition**, **RopQueryRows**, **RopSeekRow**, **RopSeekRowFractional**, or **RopSeekRowBookmark**.

3.2.4.3 Initializing Asynchronous RPC Notifications

The client SHOULD<19> support the use of asynchronous RPCs as means to notify the client of pending notifications. To initialize asynchronous RPC notifications on the server, the client sends the **EcDoAsyncConnectEx** method, as specified in [MS-OXCRPC] section 3.1.4.4, followed by the **EcDoAsyncWaitEx** method, as specified in [MS-OXCRPC] section 3.3.4.1. For more details about sending these two methods, see section 3.2.5.2 and section 3.2.5.3 respectively.

3.2.4.4 Initializing Push Notifications

As an alternate to polling, the client MAY<20> support receiving push notifications from the server. Push notifications use UDP datagrams as a means to notify the client of pending notifications. To initialize push notifications and register a callback address on the server, the client sends the **EcRRegisterPushNotification** method, as specified in section 3.2.5.4.

Clients that do not support push notifications SHOULD use either the basic polling method or the asynchronous RPC notification method.

3.2.4.5 Polling the Server for Notifications

In cases where neither push notifications nor asynchronous RPC notifications are being used, and the client is not actively calling the **EcDoRpcExt2** method, as specified in [MS-OXCRPC] section 3.1.4.2, or the **Execute** request type, as specified in [MS-OXCMAPIHTTP] section 2.2.4.2, the client MUST poll the server for pending notifications. To poll the server for pending notifications, the client

MUST make **EcDoRpcExt2** method calls as specified in section 3.2.5.6, or the **Execute** request type. The **EcDoRPCExt2** method call does not require a ROP request is included in the call.

If the client is polling the server, the client SHOULD poll at a regular interval, as specified by the value of the **pcmsPollsMax** field returned on the **EcDoConnectEx** method call, as specified in [MS-OXCRPC] section 3.1.4.1.

If the client is polling the server, the client SHOULD NOT poll more frequently than the interval specified by the value of the **pcmsPollsMax** field. If the client is required to respond to notifications at a rate that is more frequent than the polling interval, then the polling method is not recommended for retrieving notifications.

3.2.5 Message Processing Events and Sequencing Rules

The following sections specify the client-side message processing events and sequencing rules for this protocol.

3.2.5.1 Sending a RopRegisterNotification ROP Request

If the client is required to receive notifications from the server, the client SHOULD send a **RopRegisterNotification** ROP (section 2.2.1.2.1) message to the server to subscribe to notifications. The client MUST provide specific details about the notifications it needs to receive and the scope of the notifications, as specified in section 2.2.1.2.1. Upon receiving the **RopRegisterNotification** ROP response from the server, the client MUST save the returned handle to the Notification Subscription object. When the client no longer needs to receive notifications, the handle of the Notification Subscription object MUST be released by using the **RopRelease** ROP ([MS-OXCROPS] section 2.2.15.3).

The client can send the **RopRegisterNotification** ROP message multiple times to the server.

3.2.5.2 Sending an EcDoAsyncConnectEx Method Call

The client sends the **EcDoAsyncConnectEx** method, as specified in [MS-OXCRPC] section 3.1.4.4, to initialize the server for asynchronous RPC notifications.

The client SHOULD determine whether the server supports the **EcDoAsyncConnectEx** method by examining the server version information that is returned from the **EcDoConnectEx** method call, as specified in [MS-OXCRPC] section 3.1.4.1. For details about which minimum server version is required for using the asynchronous RPC notification method, see section 1.7.

The client can call the **EcDoAsyncConnectEx** method after a successful **EcDoConnectEx** method call. The client MUST save the returned asynchronous context handle after the **EcDoAsyncConnectEx** method call completes. The client MUST use the asynchronous context handle in the subsequent **EcDoAsyncWaitEx** method calls to the server, as specified in [MS-OXCRPC] section 3.3.4.1.

3.2.5.3 Sending an EcDoAsyncWaitEx Method Call

The client determines whether the server supports the asynchronous RPC notification method by examining the server version information that is returned from the **EcDoConnectEx** method call, as described in [MS-OXCRPC] section 3.1.4.1. To determine which minimum server version is required for using the asynchronous RPC notification method, see section 1.7.

If the server supports asynchronous RPC notifications, and the client successfully created asynchronous context handle by calling the **EcDoAsyncConnectEx** method, as specified in [MS-

OXCRPC] section 3.1.4.4, the client SHOULD call the **EcDoAsyncWaitEx** method, as specified in [MS-OXCRPC] section 3.3.4.1, to determine whether notifications are pending on the server.

For more details about receiving an **EcDoAsyncWaitEx** method response from the server, see section 3.2.5.5.1.

3.2.5.4 Sending an EcRRegisterPushNotification Method Call

The client MAY<21> make an **EcRRegisterPushNotification** method call, as specified in [MS-OXCRPC] section 3.1.4.5, to register a callback address for the session context with the server. The callback address is required in order to receive push notification UDP datagrams from the server. In addition to the callback address, the client MUST provide a buffer of opaque data to the server.

The client can register a variety of different callback address types if the server supports the address type. It is recommended — but not required — that a client register a callback address by using an address type that corresponds to the protocol being used to communicate with the server. For example, if the client makes an RPC call to **EcDoConnectEx,** as specified in [MS-OXCRPC] section 3.1.4.1, by using the TCP/IP protocol, it registers an AF_INET callback address in the **EcRRegisterPushNotification** method call.

Because of network conditions such as firewalls or the use of RPC/HTTP connections by the client, it is not always possible for the UDP datagram that is sent from the server to the client's callback address to be successful. To overcome this problem, the client SHOULD poll the server by using the polling method, even after registering a callback address with the server through an **EcRRegisterPushNotification** method call, until it receives a UDP datagram from the server. When the client receives a UDP datagram from the server at the specified callback address, it SHOULD stop polling the server and rely on datagrams pushed from the server to know when to call the **EcDoRpcExt2** method, as specified in [MS-OXCRPC] section 3.1.4.2, to retrieve event information.

3.2.5.5 Receiving Pending Notifications

This section specifies the following actions performed by the client to receive pending notifications:

- Receiving the **RopPending** ROP response (section 2.2.1.3.4).
- Receiving push notification UDP datagrams.
- Sending and receiving asynchronous RPC calls.

3.2.5.5.1 Sending and Receiving EcDoAsyncWaitEx Method Calls

When a call to the **EcDoAsyncWaitEx** method completes, as specified in [MS-OXCRPC] section 3.3.4.1, the client MUST examine its return value and the value of the **pulFlagsOut** field. If the return value is 0x00000000 and bit 0x00000001 is set in the **pulFlagsOut** field, the client SHOULD make **EcDoRpcExt2** method calls, as specified in [MS-OXCRPC] section 3.1.4.2, to receive notification details from the server. After the successful results of the **EcDoAsyncWaitEx** method call are processed, the client SHOULD make another **EcDoAsyncWaitEx** method call to continue to listen for more notifications.

If the **EcDoAsyncWaitEx** method returns a non-zero result code, it indicates that an error occurred. In this case, the client SHOULD NOT retry an **EcDoAsyncWaitEx** method call, and SHOULD instead use the push notification method specified in section 3.2.4.4. If the push notification method is not supported, the client SHOULD instead use the polling method specified in section 3.2.4.5.

3.2.5.5.2 Receiving Push Notification UDP Datagrams

Upon receiving a UDP datagram on the callback address that was previously registered by the client by means of the **EcRRegisterPushNotification** method, as specified in [MS-OXCRPC] section 3.1.4.5, the client MUST verify that the content of the UDP datagram is valid by matching it with the content of the opaque data **binary large object (BLOB)** that was provided to the server by means of the **EcRRegisterPushNotification** method. If the content of the UDP datagram is valid, the client SHOULD make **EcDoRpcExt2** method calls, as specified in [MS-OXCRPC] section 3.1.4.2, to receive notification details from the server. Otherwise, the client MUST NOT take any actions on the UDP datagram.

3.2.5.5.3 Receiving the RopPending ROP

Upon receiving the **RopPending** ROP response (section 2.2.1.3.4) either in the buffer of the **EcDoRpcExt2** method response, as specified in [MS-OXCRPC] section 3.1.4.2, or in the **Execute** request type response body,<22> as specified in [MS-OXCMAPIHTTP] section 2.2.4.2.2, the client MUST determine whether the session index provided in the **RopPending** ROP response matches any of the sessions created by the client. If the session index matches, the client SHOULD make an **EcDoRpcExt2** method call or send an **Execute** request type to receive notification details from the server by using the session context handle that is associated with the session specified by the session index. If the session index in **RopPending** ROP does not match the index of any session created by the client, the client MUST NOT take any actions.

3.2.5.6 Sending an EcDoRpcExt2 Method Call

The client can send the **EcDoRpcExt2** method, as specified in [MS-OXCRPC] section 3.1.4.2, as part of client polling as specified in section 3.2.4.5 with no ROP request operation included in the method call. Or, the client can send the **EcDoRpcExt2** method as part of a communication pattern unrelated to notifications. In either case, if any pending notifications exist on the server, the client receives either a **RopNotify** (section 2.2.1.4.1) or **RopPending** ([MS-OXCROPS] section 2.2.14.3) ROP in response to their **EcDoRpcExt2** method call, as specified in section 3.1.5.5.

3.2.5.7 Receiving Notification Details By Using the RopNotify ROP

After the client is notified of pending notifications by any of the methods described in section 3.2.5.5 the client calls the **EcDoRpcExt2** method, as described in [MS-OXCRPC] section 3.1.4.2, or sends an **Execute** request type<23>, as specified in [MS-OXCMAPIHTTP] section 2.2.4.2, to retrieve the notification details. In response to the **EcDoRpcExt2** method or the **Execute** request, the client receives a **RopNotify** ROP response (section 2.2.1.4.1).

Upon receiving a **RopNotify** ROP response, the client MUST verify that the value of the **NotificationHandle** field is a valid handle to a notification subscription or a **Table object** that was previously created by the client. If the value of the **NotificationHandle** field is valid, the client can update its internal state by using the details provided in the **RopNotify** ROP response. Otherwise, the client MUST ignore the **RopNotify** ROP response.

When the client subscribes to **TableModified** event notifications, the client MUST NOT make any assumptions about the level of notifications that it will receive and the client MUST be able to handle any of the three response types specified in section 3.1.4.3.

3.2.6 Timer Events

None.

3.2.7 Other Local Events

None.

4 Protocol Examples

The examples in this section are XML fragments that contain various notifications. The type of notification in each case is identified by the **name** attribute of the **Data** element.

[XML]

```
<Data name="NewMailNotification">
    <Buffer>
        02                  // NotificationType is NewMail
        80                  // Message
        010000000078291F    // New message FolderId
        0100000000783484    // New message MessageId

        22000000            // MessageFlags
        00                  // UnicodeFlag indicates ASCII
        49504D2E4E6F746500  // MessageClass
    </Buffer>
</Data>

<Data name="ObjectCreatedNotification">
    <Buffer>
        04                  // NotificationType is ObjectCreated
        00                  // No flags
        0100000000782781    // New object FolderId
        0100000000782780    // Parent FolderId
        0000                // TagCount
    </Buffer>
</Data>

<Data name="ObjectCreatedNotification">
    <Buffer>
        04                  // NotificationType is ObjectCreated
        80                  // Message
        0100000000782780    // New message FolderId
        0100000000784172    // New message MessageId

        1F00                // TagCount
        0B001B0E            // Tags
        0300790E
        02010B30
        0300A166
        0300F13F
        40000730
        40000830
        0201F93F
        1E00F83F
        03005940
        0201FB3F
        1E00FA3F
        03005A40
        0201BD67
        0201BE67
        40000967
        1F003510
        1F000010
        02010910
```

```
        02011310
        1E00040E
        1E00030E
        1F003700
        1F003D00
        1F001D0E
        0B001F0E
        0300FD3F
        40003900
        4000060E
        0300080E
        0300230E
    </Buffer>
</Data>

<Data name="ObjectDeletedNotification">
    <Buffer>
        08                  // NotificationType is ObjectDeleted
        00                  // No flags
        0100000000782780    // FolderId
        010000000078277F    // ParentFolderId
    </Buffer>
</Data>

<Data name="ObjectModifiedNotification">
    <Buffer>
        10                  // NotificationType is ObjectModified
        00                  // No flags
        0100000000782780    // FolderId

        0200                // TagCount
        03003866            // Tags
        0B000A36
    </Buffer>
</Data>

<Data name="ObjectModifiedNotification">
    <Buffer>
        10                  // NotificationType is ObjectModified
        20                  // U bitmask, unread items changed
        010000000078291F    // FolderId
        0100                // TagCount
        03000336            // Tags
        00000000            // Unread message count
    </Buffer>
</Data>

<Data name="ObjectModifiedNotification">
    <Buffer>
        10                  // NotificationType is ObjectModified
        10                  // T bitmask, total items changed
        0100000000782780    // FolderId

        0400                // TagCount
        03000236            // Tags
        0300080E
        0300AF66
        0300B366
```

```
        01000000            // TotalMessageCount
    </Buffer>
</Data>

<Data name="ObjectModifiedNotification">
    <Buffer>
        10                  // NotificationType is ObjectModified
        30                  // U bitmask, unread items changed
        010000000078291F    // FolderId

        0500                // TagCount
        03000236            // Tags
        03000336
        0300080E
        0300AF66
        0300B366

        04000000            // TotalMessageCount
        03000000            // UnreadMessageCount
    </Buffer>
</Data>

<Data name="ObjectMovedNotification">
    <Buffer>
        20                  // NotificationType isObjectMoved
        80                  // Message
        0100000000782781    // Message FolderId
        0100000000784378    // MessageId
        0100000000782780    // OldFolderId
        0100000000784172    // OldMessageId
    </Buffer>
</Data>

<Data name="ObjectCopiedNotification">
    <Buffer>
        40                  // NotificationType is ObjectCopied
        80                  // Message
        0100000000782780    // Message FolderId
        0100000000784173    // MessageId
        0100000000782780    // OldMessageId
        0100000000784172    // OldFolderId
    </Buffer>
</Data>

<Data name="TableModifiedNotification">
    <Buffer>
        00 01       // NotificationType is TableModified
        01 00       // TableEventType is TableChanged
    </Buffer>
</Data>

<Data name="TableModifiedNotification">
    <Buffer>
        00 01       // NotificationType is TableModified
        07 00       // TableEventType is TableRestrictionChanged
    </Buffer>
</Data>

<Data name="TableRowAddModifiedNotification">
```

```
    <Buffer>
        00 01       // NotificationType is TableModified
        03 00       // TableEventType is TableRowAdded
        01 00 00 02 81 6C EA 9D // TableRowFolderID
        01 00 00 02 81 6C EA 9E // InsertAfterTableRowFolderID

        A3 00 // TableRowDataSize

        // TableRowData
        00          // No errors
        42 00 69 00 6c 00 6c 00
        79 00 20 00 44 00 2e 00
        53 00 2e 00 20 00 50 00
        72 00 6f 00 78 00 79 00 00

        00 7e
        00 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 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 49 4e 49 53
        54 52 41 54 49 56 45 20 47 52 4f 55 50 20 28 46
        59 44 49 42 4f 48 46 32 33 53 50 44 4c 54 29 2f
        43 4e 3d 52 45 43 49 50 49 45 4e 54 53 2f 43 4e
        3d 44 53 50 52 4f 58 59 00
    </Buffer>
</Data>

<Data name="TableRowAddModifiedNotification">
    <Buffer>
        00 C1       // NotificationType is TableModified and the
                    // S and M flags are set
        03 00       // TableEventType is TableRowAdded
        01 00 00 00 00 78 60 45 // FolderId
        01 00 00 02 81 6C FC 84 // MessageId
        01 00 00 00 // TableRowInstance
        01 00 00 00 00 78 60 45 // InsertAfterTableRowFolderId
        01 00 00 02 81 6C FC 82 // InsertAfterTableRowID
        01 00 00 00 // InsertAfterTableRowInstance

        A3 00 // TableRowDataSize

        // TableRowData
        00          // No errors
        42 00 69 00 6c 00 6c 00
        79 00 20 00 44 00 2e 00
        53 00 2e 00 20 00 50 00
        72 00 6f 00 78 00 79 00 00

        00 7e
        00 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 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 49 4e 49 53
        54 52 41 54 49 56 45 20 47 52 4f 55 50 20 28 46
        59 44 49 42 4f 48 46 32 33 53 50 44 4c 54 29 2f
        43 4e 3d 52 45 43 49 50 49 45 4e 54 53 2f 43 4e
        3d 44 53 50 52 4f 58 59 00
```

```
    </Buffer>
</Data>

<Data name="TableRowAddModifiedNotification">
    <Buffer>
        00 01       // NotificationType is TableModified
        05 00       // TableEventType is TableRowModified
        01 00 00 00 00 78 60 45 // FolderId
        01 00 00 00 00 78 60 50 // InsertAfterTableRowFolderID

        A3 00 // TableRowDataSize

        // TableRowData
        00          // No errors
        42 00 69 00 6c 00 6c 00
        79 00 20 00 44 00 2e 00
        53 00 2e 00 20 00 50 00
        72 00 6f 00 78 00 79 00 00

        00 7e
        00 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 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 49 4e 49 53
        54 52 41 54 49 56 45 20 47 52 4f 55 50 20 28 46
        59 44 49 42 4f 48 46 32 33 53 50 44 4c 54 29 2f
        43 4e 3d 52 45 43 49 50 49 45 4e 54 53 2f 43 4e
        3d 44 53 50 52 4f 58 59 00
    </Buffer>
</Data>

<Data name="TableRowAddModifiedNotification">
    <Buffer>
        00 C1       // NotificationType is TableModified and the
                    // S and M flags are set
        05 00       // TableEventType is TableRowModified
        01 00 00 00 00 78 60 45 // TableRowFolderID
        01 00 00 02 81 6C FC 83 // TableRowMessageID
        01 00 00 00 // TableRowInstance
        01 00 00 00 00 78 60 46 // InsertAfterTableRowFolderID
        01 00 00 02 81 6C FC 84 // InsertAfterTableRowID
        01 00 00 00 // InsertAfterTableRowInstance

        A3 00 // TableRowDataSize

        // TableRowData
        00          // No errors
        42 00 69 00 6c 00 6c 00
        79 00 20 00 44 00 2e 00
        53 00 2e 00 20 00 50 00
        72 00 6f 00 78 00 79 00 00

        00 7e
        00 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 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 49 4e 49 53
```

```
        54 52 41 54 49 56 45 20 47 52 4f 55 50 20 28 46
        59 44 49 42 4f 48 46 32 33 53 50 44 4c 54 29 2f
        43 4e 3d 52 45 43 49 50 49 45 4e 54 53 2f 43 4e
        3d 44 53 50 52 4f 58 59 00
    </Buffer>
</Data>

<Data name="TableRowDeletedModifiedNotification">
    <Buffer>
        00 01       // NotificationType is TableModified
        04 00       // TableEventType is TableRowDeleted
        01 00 00 00 00 78 60 45 // FolderId
    </Buffer>
</Data>

<Data name="TableRowDeletedModifiedNotification">
    <Buffer>
        00 C1       // NotificationType is TableModified and the
                    // S and M flags are set
        04 00       // TableEventType is TableRowDeleted
        01 00 00 02 81 6C EA 96 // TableRowFolderID
        01 00 00 02 81 6D 09 01 // TableRowMessageID
        01 00 00 00 // TableRowInstance
    </Buffer>
</Data>
```

5 Security

5.1 Security Considerations for Implementers

There are no special security considerations specific to this protocol. However, general security considerations pertaining to the underlying ROP transport protocol described in [MS-OXCROPS] do apply to this protocol.

5.2 Index of Security Parameters

None.

6 Appendix A: Product Behavior

The information in this specification is applicable to the following Microsoft products or supplemental software. References to product versions include released service packs:

- Microsoft Exchange Server 2003
- Microsoft Exchange Server 2007
- Microsoft Exchange Server 2010
- Microsoft Exchange Server 2013
- Microsoft Office Outlook 2003
- Microsoft Office Outlook 2007
- Microsoft Outlook 2010
- Microsoft Outlook 2013

Exceptions, if any, are noted below. If a service pack or Quick Fix Engineering (QFE) number appears with the product version, behavior changed in that service pack or QFE. The new behavior also applies to subsequent service packs of the product unless otherwise specified. If a product edition appears with the product version, behavior is different in that product edition.

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

<1> Section 2.1: Exchange 2003, Exchange 2007, Exchange 2010, the initial release of Exchange 2013, Office Outlook 2003, Office Outlook 2007, Outlook 2010, and the initial release of Outlook 2013 do not support the MAPI extensions to HTTP. The MAPI extensions to HTTP were introduced in Outlook 2013 SP1 and Exchange 2013 SP1.

<2> Section 2.2.1: Exchange 2003, Exchange 2007, Exchange 2010, the initial release of Exchange 2013, Office Outlook 2003, Office Outlook 2007, Outlook 2010, and the initial release of Outlook 2013 do not support the **NotificationWait** request type. The **NotificationWait** request type was introduced in Outlook 2013 SP1 and Exchange 2013 SP1.

<3> Section 2.2.1.3.1: Exchange 2003 and Office Outlook 2003 do not support the **EcDoAsyncConnectEx** method, as specified in [MS-OXCRPC] section 3.1.4.4.

<4> Section 2.2.1.3.2: Exchange 2003 and Office Outlook 2003 do not support the **EcDoAsyncWaitEx** method, as specified in [MS-OXCRPC] section 3.3.4.1.

<5> Section 2.2.1.3.3: Exchange 2003, Exchange 2007, Office Outlook 2003, Office Outlook 2007, and Outlook 2010 support the **EcRRegisterPushNotification** method, as specified in [MS-OXCRPC] section 3.1.4.5. The initial release version of Exchange 2010, and Exchange 2010 SP1, do not support the **EcRRegisterPushNotification** method, and the returned value is always **ecNotSupported** (0x80040102). Exchange 2010 SP2 does support the **EcRRegisterPushNotification** method if a registry key is created to support push notifications, as described in [MSFT-ConfigStaticUDPPort]. Outlook 2013 does not send the **EcRRegisterPushNotification** RPC method call. Exchange 2013 does not support the **EcRRegisterPushNotification** method, and the returned value is always **ecNotSupported** (0x800401012).

<6> Section 2.2.1.3.4: Exchange 2003, Exchange 2007, Exchange 2010, the initial release of Exchange 2013, Office Outlook 2003, Office Outlook 2007, Outlook 2010, and the initial release of Outlook 2013 do not support the **Execute** request type. The **Execute** request type was introduced in Outlook 2013 SP1 and Exchange 2013 SP1.

<7> Section 2.2.1.4.1: Exchange 2003, Exchange 2007, Exchange 2010, the initial release of Exchange 2013, Office Outlook 2003, Office Outlook 2007, Outlook 2010, and the initial release of Outlook 2013 do not support the **Execute** response type. The **Execute** response type was introduced in Outlook 2013 SP1 and Exchange 2013 SP1.

<8> Section 2.2.1.4.1.1: Exchange 2003, Exchange 2007, and Exchange 2010 do not set the value of the **TagCount** field to 0x0000; they set the value of the field to the number of property tags in the **Tags** field.

<9> Section 2.2.1.4.1.1: Exchange 2003, Exchange 2007, Exchange 2010, the initial release of Exchange 2013, Office Outlook 2003, Office Outlook 2007, Outlook 2010, and the initial release of Outlook 2013 do not support the **Execute** request type. The **Execute** request type was introduced in Outlook 2013 SP1 and Exchange 2013 SP1.

<10> Section 3.1.1: Exchange 2003, Exchange 2007, Exchange 2010, the initial release of Exchange 2013, Office Outlook 2003, Office Outlook 2007, Outlook 2010, and the initial release of Outlook 2013 do not support the **session context cookie**. The **session context cookie** was introduced in Outlook 2013 SP1 and Exchange 2013 SP1.

<11> Section 3.1.4.3: Exchange 2003 and Exchange 2007 do not require that a table view is created in order to send table notifications.

<12> Section 3.1.4.3: Exchange 2003 and Exchange 2007 do not stop sending notifications if the **RopResetTable** ROP ([MS-OXCROPS] section 2.2.5.15) is received.

<13> Section 3.1.5.2: Exchange 2003 does not support the **EcDoAsyncConnectEx** method call or asynchronous RPC notifications.

<14> Section 3.1.5.3: Exchange 2003 does not support the **EcDoAsyncWaitEx** method call or asynchronous RPC notifications.

<15> Section 3.1.5.4: Exchange 2003 and Exchange 2007 support push notifications and the **EcRRegisterPushNotification** method, as specified in [MS-OXCRPC] section 3.1.4.5. The initial release version of Exchange 2010, and Exchange 2010 SP1 do not support push notifications or the **EcRRegisterPushNotification** method. Exchange 2010 SP2 does support push notifications and the **EcRRegisterPushNotification** method if a registry key is created, as described in [MSFT-ConfigStaticUDPPort]. Exchange 2013 does not support push notifications of the **EcRRegisterPushNotification** method.

<16> Section 3.1.5.6: Exchange 2007 sends a **RopPending** ROP response (section 2.2.1.3.4) to the client whenever there are pending notifications on any linked session contexts.

<17> Section 3.2.4.2: Exchange 2003 and Exchange 2007 do not require that the client send any ROPs to the server in order to receive **TableModified** event notifications, as specified in section 2.2.1.1.1. In Exchange 2003 and Exchange 2007, the subscription is created automatically when the client creates a Table object on the server.

<18> Section 3.2.4.2: The client will continue to receive table notifications even if the **RopResetTable** ROP ([MS-OXCROPS] section 2.2.5.15) is sent, if the server is Exchange 2003 or Exchange 2007.

<19> Section 3.2.4.3: Office Outlook 2003 does not support the asynchronous RPC notification method. Office Outlook 2007, Outlook 2010, and Outlook 2013 do support the asynchronous RPC notification method.

<20> Section 3.2.4.4: Office Outlook 2003, Office Outlook 2007 and Outlook 2010 do support the push notification method. Outlook 2013 does not support the push notification method.

<21> Section 3.2.5.4: Outlook 2013 does not support the push notification method and does not send the **EcRRegisterPushNotification** RPC method call.

<22> Section 3.2.5.5.3: Exchange 2003, Exchange 2007, Exchange 2010, the initial release of Exchange 2013, Office Outlook 2003, Office Outlook 2007, Outlook 2010, and the initial release of Outlook 2013 do not support the **Execute** request type. The **Execute** request type was introduced in Outlook 2013 SP1 and Exchange 2013 SP1.

<23> Section 3.2.5.7: Exchange 2003, Exchange 2007, Exchange 2010, the initial release of Exchange 2013, Office Outlook 2003, Office Outlook 2007, Outlook 2010, and the initial release of Outlook 2013 do not support the **Execute** request type. The **Execute** request type was introduced in Outlook 2013 SP1 and Exchange 2013 SP1.

7 Change Tracking

No table of changes is available. The document is either new or has had no changes since its last release.

8 Index