

[MS-OXCNOTIF]: Core Notifications Protocol Specification

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's Open Specification Promise (available here: <http://www.microsoft.com/interop/osp>) or the Community Promise (available here: <http://www.microsoft.com/interop/cp/default.msp>). 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 iplq@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.
- **Fictitious Names.** The example companies, organizations, products, domain names, e-mail addresses, logos, people, places, and events depicted in this documentation are fictitious. No association with any real company, organization, product, domain name, email address, logo, person, place, or event is intended or should be inferred.

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

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

Revision Summary

Date	Revision History	Revision Class	Comments
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.

Table of Contents

1	Introduction	5
1.1	Glossary	5
1.2	References.....	6
1.2.1	Normative References.....	6
1.2.2	Informative References	6
1.3	Overview	6
1.3.1	Pending Notifications	6
1.3.1.1	RopPending.....	7
1.3.1.2	Polling.....	7
1.3.1.3	Push Notification.....	7
1.3.1.4	Asynchronous RPC Notification	7
1.3.2	Notification Details	7
1.4	Relationship to Other Protocols.....	7
1.5	Prerequisites/Preconditions	8
1.6	Applicability Statement.....	8
1.7	Versioning and Capability Negotiation.....	8
1.8	Vendor-Extensible Fields.....	8
1.9	Standards Assignments	8
2	Messages.....	9
2.1	Transport.....	9
2.2	Message Syntax	9
2.2.1	Notifications	9
2.2.1.1	Server Event Types.....	9
2.2.1.1.1	TableModified Event Types	9
2.2.1.2	Subscription Management.....	10
2.2.1.2.1	RopRegisterNotification	10
2.2.1.2.1.1	NotificationTypes	11
2.2.1.2.2	RopSynchronizationOpenAdvisor	11
2.2.1.2.3	RopRegisterSynchronizationNotifications	12
2.2.1.2.4	RopSetSynchronizationNotificationGuid	12
2.2.1.3	Pending Notifications.....	12
2.2.1.3.1	RopPending	12
2.2.1.3.2	EcRRegisterPushNotification	12
2.2.1.3.3	EcDoAsyncConnectEx.....	13
2.2.1.3.4	EcDoAsyncWaitEx.....	13
2.2.1.4	Notification Details.....	13
2.2.1.4.1	RopNotify	13
2.2.1.4.1.1	NotificationFlags	16
2.2.1.4.1.2	TableEventType.....	17
2.2.1.4.1.3	MessageFlags.....	18
2.2.1.4.1.4	MessageClass.....	18
3	Protocol Details.....	19
3.1	Server Details	19
3.1.1	Abstract Data Model	19
3.1.2	Timers	19
3.1.3	Initialization	19
3.1.3.1	Subscribing for Notifications.....	19
3.1.3.1.1	Receiving RopRegisterNotification.....	19

3.1.3.1.2	Receiving RopSynchronizationOpenAdvisor	19
3.1.3.1.3	Receiving RopRegisterSynchronizationNotifications	19
3.1.3.1.4	Receiving RopSetSynchronizationNotificationGuid	19
3.1.3.1.5	Subscribing for Table Notifications	20
3.1.3.2	Initializing Pending Notifications	20
3.1.3.2.1	Receiving EcRRegisterPushNotification	20
3.1.3.2.2	Receiving EcDoAsyncConnectEx	20
3.1.4	Higher-Layer Triggered Events	20
3.1.5	Message Processing Events and Sequencing Rules	21
3.1.5.1	Notifying Client about Pending Notifications	21
3.1.5.1.1	Sending RopPending	21
3.1.5.1.2	Sending Push Notification Datagram	21
3.1.5.1.3	Receiving and Completing Asynchronous RPC call	21
3.1.5.2	Sending Notification Details	21
3.1.5.2.1	Sending RopNotify	21
3.1.6	Timer Events	21
3.1.7	Other Local Events	22
3.2	Client Details	22
3.2.1	Abstract Data Model	22
3.2.2	Timers	22
3.2.3	Initialization	22
3.2.3.1	Subscribing for Notifications	22
3.2.3.1.1	Sending RopRegisterNotification	22
3.2.3.1.2	Sending RopSynchronizationOpenAdvisor	22
3.2.3.1.3	Sending RopRegisterSynchronizationNotifications	22
3.2.3.1.4	Sending RopSetSynchronizationNotificationGuid	22
3.2.3.1.5	Subscribing for Table Notifications	23
3.2.3.2	Initializing Push Notifications	23
3.2.3.2.1	Sending EcRRegisterPushNotifications	23
3.2.3.2.2	Sending EcDoAsyncConnectEx	23
3.2.4	Higher-Layer Triggered Events	23
3.2.5	Message Processing Events and Sequencing Rules	24
3.2.5.1	Receiving Notification About Pending Notifications	24
3.2.5.1.1	Receiving RopPending	24
3.2.5.1.2	Receiving Push Notification Datagram	24
3.2.5.1.3	Sending and Receiving EcDoAsyncWaitEx	24
3.2.5.2	Receiving Notification Details	24
3.2.5.2.1	Receiving RopNotify	24
3.2.6	Timer Events	24
3.2.7	Other Local Events	24
4	Protocol Examples	25
5	Security	31
5.1	Security Considerations for Implementers	31
6	Appendix A: Product Behavior	32
7	Change Tracking	33
8	Index	34

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 contain RFC 2119 language. Sections 1.5 and 1.9 are also normative but cannot contain RFC 2119 language. All other sections and examples in this specification are informative.

1.1 Glossary

The following terms are defined in [\[MS-GLOS\]](#):

ASCII
GUID
handle
Hypertext Transfer Protocol (HTTP)
remote procedure call (RPC)
Unicode

The following terms are defined in [\[MS-OXGLOS\]](#):

asynchronous context handle
Logon object
remote operation (ROP)
ROP request buffer
ROP response buffer
Server object
Server object handle table
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.

ICS Advisor object: A handle to an object that is created on a server and that receives event notifications.

Internet datagram: A unit of data that is exchanged between a pair of Internet modules, including the Internet header.

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

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. Please check the archive site, <http://msdn2.microsoft.com/en-us/library/E4BD6494-06AD-4aed-9823-445E921C9624>, as an additional source.

[MS-OXCFOOLD] Microsoft Corporation, "[Folder Object Protocol Specification](#)".

[MS-OXCMSG] Microsoft Corporation, "[Message and Attachment Object Protocol Specification](#)".

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

[MS-OXCRPC] Microsoft Corporation, "[Wire Format Protocol Specification](#)".

[MS-OXCSTOR] Microsoft Corporation, "[Store Object Protocol Specification](#)".

[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>

1.2.2 Informative References

[MSDN-ENM] Microsoft Corporation, "Event Notification in MAPI", [http://msdn.microsoft.com/en-us/library/ms528269\(EXCHG.10\).aspx](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](http://msdn.microsoft.com/en-us/library/ms740673(VS.85).aspx)

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

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

1.3 Overview

The messaging client can register to receive notifications about specific events that can occur on the messaging server. When an event occurs on the server and a client has registered to receive the notification, the server sends the notification details to the client in the **ROP response buffer** on the **EcDoRpcExt2** calls, as specified in [\[MS-OXCRPC\]](#) section 3.1.4.12, in the format described by the **ROPNotify remote operation (ROP)** ([\[MS-OXCROPS\]](#) section 2.2.14.5).

The Core Notifications Protocol is logically divided into two parts: one that notifies a client about pending notifications, and one that transmits the notifications. The following subsections describe these two components of the protocol.

1.3.1 Pending Notifications

Because the receipt of notification details is done only through the ROP response buffer that is returned from **EcDoRpcExt2** ([\[MS-OXCRPC\]](#) section 3.1.4.12) calls, the server needs a mechanism to inform the client of any pending notifications on the session context on the server when the client is idle and not actively calling the **EcDoRpcExt2** method. The server provides four different methods that a client can use to be notified of pending notifications.

The following subsections describe the four methods that the server provides.

1.3.1.1 RopPending

If there are pending notifications for the session, the server sends the **RopPending** ROP ([\[MS-OXCROPS\]](#) section 2.2.14.6) in the ROP response buffer on the **EcDoRpcExt2** call, as specified in [\[MS-OXCRPC\]](#) section 3.1.4.12.

1.3.1.2 Polling

If a client is idle and is not making **EcDoRpcExt2** calls, it cannot receive the **RopNotify** ROP. The simplest way for a client to retrieve notification details is to make **EcDoRpcExt2** calls at regular intervals. The server enables the client to call the **EcDoRpcExt2** method with no ROP request operations. This provides the client a means to retrieve any pending notifications.

The interval at which the client polls the server for notifications is returned on the **EcDoConnectEx** call as specified in [\[MS-OXCRPC\]](#) section 3.1.4.11. The output parameter *pcmsPollsMax* in both of these calls contains the number of milliseconds that the client waits before polling the server for event information. It is not recommended that the client poll the server more frequently than what is returned by the server. If the client needs to be very responsive to events on the server, the polling method is not recommended.

1.3.1.3 Push Notification

Instead of polling the server at regular intervals to get notification details, the client can register a **callback address** with the server. The server will send an **Internet datagram** to the callback address to inform the client that notifications are pending on the server for the session.

Clients connecting by means of the **RPC/HTTP** protocol can use the push notification method of being notified of pending notifications. [<1>](#)

1.3.1.4 Asynchronous RPC Notification

The **Asynchronous RPC Notification** method enables the client to make an asynchronous remote procedure call (RPC) call to the server if the server does not complete the RPC call until there is a notification for the session. This method works through RPC/HTTP protocol connections with the server in cases where the **Push Notification** method will not work. The client determines whether the server supports this notification method by examining the server version information that is returned from the **EcDoConnectEx** call. To determine which minimum server version is required for using the asynchronous RPC notification method, see section [1.7](#).

1.3.2 Notification Details

After the client is notified of pending notifications by any of the methods described in sections [1.3.1.1](#), [1.3.1.2](#), [1.3.1.3](#), and [1.3.1.4](#), the client calls the **EcDoRpcExt2** method to retrieve the notification details. The server adds any notification details in the ROP response buffer of the **EcDoRpcExt2** by using the **RopNotify** response command. The server returns as many notification details through multiple **RopNotify** response commands as the ROP response buffer allows. If the server is not able to fit all pending notifications in the response buffer, the server also returns the **RopPending** response command to indicate that some notifications are still pending.

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 MAPI provider, see [\[MSDN-ENM\]](#).

This specification relies on an understanding of both [\[MS-OXCRPC\]](#) and [\[MS-OXCROPS\]](#).

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

The Core Notifications 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 [\[MS-OXCRPC\]](#), the Remote Operations (ROP) List and Encoding Protocol [\[MS-OXCROPS\]](#), and Internet protocols as specified in section [2.1](#).
- Protocol Versions: This protocol has only one interface version.
- Capability Negotiation: The protocol does not require asynchronous remote procedure call (RPC) notifications to be implemented. The client examines the server version to determine whether asynchronous RPC notifications are supported. For more details about how to determine server version, see [\[MS-OXCRPC\].<2>](#)
- Localization: This protocol passes text strings in notification details. Localization considerations for such strings are specified in section [2.2.1.4.1.4](#).

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 remote procedure call (RPC) transport, as specified in [\[MS-OXCRPC\]](#).

Internet datagrams are sent from server to client by using underlying networking protocols. For more information, see [\[MSDN-WS2\]](#).

2.2 Message Syntax

2.2.1 Notifications

2.2.1.1 Server Event Types

The following table describes the events that happen on the server. Clients can register to receive notifications about these events.

Event name	Description
CriticalError	A critical error has occurred on the server. <3>
NewMail	A new e-mail message has been received by 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.
ObjectCopied	An existing object has been copied on the server.
SearchComplete	A search operation has been completed on the server.

2.2.1.1.1 TableModified Event Types

The following table describes the table modification event types.

Event name	Description
TableChanged	A table has been changed.
TableError	An error has occurred.
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.

Event name	Description
TableSortDone	A table sort has been completed.
TableRestrictionChanged	A table restriction has been changed.
TableColumnsChanged	Table columns have been changed.

When a client subscribes to notifications about table changes, the server does one of the following three things, listed in order from the most useful to the least useful to the client:

- Generates an informative notification such as **TableRowAdded**.
- Generates a basic notification for a **TableChanged** event.
- Does not generate a notification at all.

The notification level is server implementation specific, and the client **MUST NOT** make any assumptions about the level of notifications that it will receive.

A client **MUST** be able to handle any of the three response types specified in the preceding bulleted list. The server **SHOULD** generate the most useful response that it is capable of generating, as specified in the preceding list.

2.2.1.2 Subscription Management

2.2.1.2.1 RopRegisterNotification

The **RopRegisterNotification** ROP creates a subscription for specified notifications on the server and returns a **handle** of the subscription to the client. The following table describes the parts of the **notification subscription** request.

Part name	Type	Size	Description
RopId	Byte	1	An unsigned 8-bit integer. This value specifies the type of ROP. For this operation, this field is set to "0x29".
LogonId	Byte	1	An unsigned 8-bit integer. This value specifies the logon identifier (ID) associated with this operation.
InputHandleIndex	Byte	1	An unsigned 8-bit integer. This index specifies the location in the Server object handle table where the handle for the input Server object is stored.
OutputHandleIndex	Byte	1	An unsigned 8-bit integer. This index specifies the location in the Server object handle table where the handle for the output Server object will be stored.
NotificationTypes	Byte	1	A set of bits describing notifications that the client is interested in receiving. For more details, see section 2.2.1.2.1.1 .
Reserved	Byte	1	This field is reserved. The field value MUST be "0" (zero). The server behavior is undefined if the value is not 0.
WantWholeStore	Byte	1	"TRUE" (nonzero) if the scope for notifications is the entire database; otherwise, "FALSE" (zero).
FolderID	ID	8	The ID of the folder to limit the scope of notifications. This field

Part name	Type	Size	Description
			is available only if the WantWholeStore value is "0" (zero).
MessageID	ID	8	The ID of the message inside the folder referenced by FolderID to limit the scope for notifications. This field is available only if the WantWholeStore value is "0" (zero).

The following table describes the notification subscription response.

Response name	Type	Size	Description
OutputHandleIndex	Byte	1	The handle of the notification subscription object created by this ROP. This index MUST be set to the OutputHandleIndex value specified in the request.

2.2.1.2.1.1 NotificationTypes

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

Value	Meaning
0x01	The server sends notifications to the client when CriticalError events occur within the scope of interest. <4>
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](#).

2.2.1.2.2 RopSynchronizationOpenAdvisor

The **RopSynchronizationOpenAdvisor** ROP creates an **ICS Advisor object** [<5>](#) on the server and returns a handle of the object to the client. The following table shows the ICS Advisor request.

Request name	Type	Size	Description
InputHandleIndex	Handle	1	The handle of the Logon object . For more details, see [MS-OXCROPS] section 2.2.14.2.

The following table shows the ICS Advisor response.

Response name	Type	Size	Description
OutputHandleIndex	Handle	1	The handle of the ICS Advisor object created by this ROP. For more details, see [MS-OXCROPS] section 2.2.14.2.<6>

2.2.1.2.3 RopRegisterSynchronizationNotifications

The **RopRegisterSynchronizationNotifications** ROP creates a subscription for *StatusObjectModified* notifications on the server. The following table describes the parts of the **RopRegisterSynchronizationNotifications** request.

Part name	Type	Size	Description
InputHandleIndex	Byte	1	The handle of the ICS Advisor object.
FolderCount	Short	2	The number of folder IDs that limit the scope of the notification subscription.
FolderIDs	ID[]	<i>NumberOfFolderIDs</i>	The list of folder IDs that limit the scope of the notification subscription.
FolderChangeNumbers	ULong[]	<i>NumberOfFolderIDs</i>	The list of folder change numbers (CNs).

For details about the response to this request, see [\[MS-OXCROPS\]](#) section 2.2.14.3.

2.2.1.2.4 RopSetSynchronizationNotificationGuid

The **RopSetSynchronizationNotificationGuid** ROP assigns a notification **GUID** to an ICS Advisor object on the server. The following table describes the parts of the **RopSetSynchronizationNotificationGuid** request.

Part name	Type	Size	Description
InputHandleIndex	Byte	1	The handle of the ICS Advisor object. For more details, see [MS-OXCROPS] section 2.2.14.4.
NotificationGuid	GUID	16	A notification GUID to assign to the ICS Advisor object.

For details about the response to this request, see [\[MS-OXCROPS\]](#) section 2.2.14.4.

2.2.1.3 Pending Notifications

2.2.1.3.1 RopPending

The **RopPending** ROP notifies the client that there are pending notifications on the server for the client. This ROP MUST appear only in response buffers of the **EcDoRpcExt2** method. For more details, see [\[MS-OXCROPS\]](#) section 2.2.14.6.

2.2.1.3.2 EcRRegisterPushNotification

EcRRegisterPushNotification is an RPC method that is used to register a callback address of a client on the server. See [\[MS-OXCRPC\]](#) section 3.1.4.5 for more details.

2.2.1.3.3 EcDoAsyncConnectEx

EcDoAsyncConnectEx is an RPC method that is used to acquire an **asynchronous context handle** on the server to use in subsequent **EcDoAsyncWaitEx** calls. For more details, see [\[MS-OXCRPC\]](#) section 3.1.4.15.

2.2.1.3.4 EcDoAsyncWaitEx

EcDoAsyncWaitEx is an asynchronous RPC method that is used to inform a client about pending notifications on the server. For more details, see [\[MS-OXCRPC\]](#) section 3.3.4.1.

2.2.1.4 Notification Details

2.2.1.4.1 RopNotify

The **RopNotify** ROP 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. The following table describes the parts of the **RopNotify** request.

Part name	Type	Size	Description
NotificationHandle	Handle	4	The handle of the target object for the notification. The target object can be a notification subscription, an ICS Advisor object, or a table.
NotificationFlags	UShort	2	A set of bits describing the type of the notification and the availability of the notification data fields. For details, see section 2.2.1.4.1.1 .
TableEventType	Byte	2	A subtype of the notification for a TableModified event. This field is available only if the NotificationType value in the NotificationFlags field is "0x0100". For details, see section 2.2.1.4.1.2 .
TableRowFolderID	ID	8	The folder ID of the item triggering the notification. This field is available only if the TableEventType field is available and is "0x03", "0x04", or "0x05".
TableRowMessageID	ID	8	The message ID 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 "0x03", "0x04", or "0x05".
TableRowInstance	ULong	4	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 "0x03", "0x04", or "0x05".

Part name	Type	Size	Description
			"0x05".
InsertAfterTableRowFolderID	ID	8	The old folder ID of the item triggering the notification. This field is available only if the TableEventType field is available and is "0x03" or "0x05".
InsertAfterTableRowID	ID	8	The old message ID 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 "0x03" or "0x05".
InsertAfterTableRowInstance	ULong	4	An identifier of the instance of the row where the modified row is inserted.
TableRowDataSize	UShort	2	The length of the table row data. This field is available only if the TableEventType field is available and is "0x03" or "0x05".
TableRowData	String	<i>TableRowDataSize</i>	The table row data. This field is available only if the TableEventType field is available and is "0x03" or "0x05".
HierarchyChanged	Byte	1	"TRUE" (nonzero) if the folder hierarchy has changed; otherwise, "FALSE" (zero). This field is available only if the NotificationType value in the NotificationFlags field is "0x0200".
FolderIDNumber	ULong	4	The number of a folder ID. This field is available only if the NotificationType value in the NotificationFlags field is "0x0200".
FolderIDs	GID []	<i>FolderIDNumber</i>	Folder IDs. This field is available only if the NotificationType value in the NotificationFlags field is "0x0200".
ICSChangeNumbers	ULong[]	<i>FolderIDNumber</i>	Folder change numbers. This field is available only if the NotificationType value in the NotificationFlags field is "0x0200".
FolderId	ID	8	The folder ID of the item triggering the event. This field is available only if the NotificationType value in the NotificationFlags field is not "0x0100", "0x0200", or "0x0400".

Part name	Type	Size	Description
MessageId	ID	8	The message ID of the item triggering the event. This field is available only if the NotificationType value in the NotificationFlags field is not "0x0100", "0x0200", or "0x0400", and bit "0x8000" is set in the NotificationFlags field.
ParentFolderId	ID	8	The folder ID of the parent folder of the item triggering the event. This field is available only if the NotificationType value is "0x0004", "0x0008", "0x0020", or "0x0040", and it is sent for either a message in a search folder (both bit "0x4000" and bit "0x8000" are set in the NotificationFlags field) or a folder (both bit "0x400"0 and bit "0x8000" are not set in the NotificationFlags field).
OldFolderId	ID	8	The old folder ID of the item triggering the event. This field is available only if the NotificationType value in the NotificationFlags field is "0x0020" or "0x0040".
OldMessageId	ID	8	The old message ID of the item triggering the event. This field is available only if the NotificationType value in the NotificationFlags field is "0x0020" or "0x0040" and bit "0x800"0 is set in the NotificationFlags field.
OldParentFolderId	ID	8	The old parent folder ID of the item triggering the event. This field is available only if the NotificationType value in the NotificationFlags field is "0x002"0 or "0x0040" and bit "0x8000" is not set in the NotificationFlags field.
TagCount	UShort	2	The number of property tags. This field is available only if the NotificationType value in the NotificationFlags field is "0x0004" or "0x0010". A value of "0xFFFF" is returned if there were too many tags to fit into the response and the list of property tags was omitted.
Tags	ULong[]	<i>TagCount</i>	The list of IDs of properties that have changed. This field is available only if the TagCount field is available and the TagCount value is not "0x00" or "0xFFFF".

Part name	Type	Size	Description
TotalMessageCount	ULong	4	The total number of items in the folder triggering this event. This field is available only if bit "0x100" is set in the NotificationFlags field.
UnreadMessageCount	ULong	4	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	ULong	4	The message flags of new mail that has been received. This field is available only if the NotificationType value in the NotificationFlags field is "0x0002".
UnicodeFlag	Byte	1	"TRUE" (nonzero) if the MessageClass value is in Unicode ; otherwise, "FALSE" (zero). This field is available only if the NotificationType value in the NotificationFlags field is "0x0002".<7>
MessageClass	String	<i>Variable</i>	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" (nonzero). The string is in ASCII if UnicodeFlag is set to "FALSE" (zero). This field is available only if the NotificationType value in the NotificationFlags field is "0x0002".

2.2.1.4.1.1 NotificationFlags

NotificationFlags is a 16-bit combination of an enumeration and flags. The layout is shown in the following table.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
NotificationType												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
0x0001	The notification is for a CriticalError event.

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.
0x0200	The notification is for a StatusObjectModified event.
0x0400	This value is reserved.

T (1 bit): Value = 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): Value = 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): Value = 0x4000. The notification is caused by an event in a search folder. If this bit is set, bit "0x800"0 MUST be set.

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

2.2.1.4.1.2 TableEventType

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

Value	Meaning
0x0001	The notification is for TableChanged events.
0x0002	The notification is for TableError events.
0x0003	The notification is for TableRowAdded events.
0x0004	The notification is for TableRowDeleted events.
0x0005	The notification is for TableRowModified events.
0x0006	The notification is for TableSortDone events.
0x0007	The notification is for TableRestrictionChanged events.
0x0008	The notification is for TableColumnsChanged events.

2.2.1.4.1.3 MessageFlags

For details, see [\[MS-OXCMSG\]](#) section 2.2.1.6.

2.2.1.4.1.4 MessageClass

For details, see [\[MS-OXCMSG\]](#) section 2.2.1.3.

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.

3.1.2 Timers

The server SHOULD allow for a certain time interval between 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 datagrams.

3.1.3 Initialization

3.1.3.1 Subscribing for Notifications

3.1.3.1.1 Receiving RopRegisterNotification

When a **RopRegisterNotification** (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** fields for future use.

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

3.1.3.1.2 Receiving RopSynchronizationOpenAdvisor

When a **RopSynchronizationOpenAdvisor** message is received by the server, the server SHOULD create a new ICS Advisor object and associate it with the session context.

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

3.1.3.1.3 Receiving RopRegisterSynchronizationNotifications

When a **RopRegisterSynchronizationNotifications** message is received by the server, the *InputHandle* parameter MUST be a valid handle of the ICS Advisor object.

The server SHOULD allow multiple **RopRegisterSynchronizationNotifications** messages to be received for the same ICS Advisor object.

The server SHOULD adjust the scope of the notification subscription with the details provided by the last **RopRegisterSynchronizationNotifications** message that was successfully processed.

3.1.3.1.4 Receiving RopSetSynchronizationNotificationGuid

When a **RopSetSynchronizationNotificationGuid** message is received by the server, the *InputHandle* parameter MUST be a valid handle of the ICS Advisor object.

The server SHOULD allow multiple **RopSetSynchronizationNotificationGuid** messages to be received for the same ICS Advisor object.

The server SHOULD assign the ICS Advisor object a notification GUID provided by the last **RopSetSynchronizationNotificationGuid** message that was successfully processed.

The server MUST NOT send any **StatusObjectModified** notifications to the client, if these notifications were triggered by a client logon that has a **PidTagChangeNotificationGuid** property value that matches the GUID assigned to the ICS Advisor object by the **RopSetSynchronizationNotificationGuid** message. For more details, see [\[MS-OXCSTOR\]](#).

3.1.3.1.5 Subscribing for Table Notifications

The server SHOULD NOT require any additional actions for a client to register for notifications on table events. If a table is created on the server, the server SHOULD create a subscription to table 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, see [\[MS-OXCFOLD\]](#).

3.1.3.2 Initializing Pending Notifications

3.1.3.2.1 Receiving EcRRegisterPushNotification

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 datagram containing the client's opaque data.

3.1.3.2.2 Receiving EcDoAsyncConnectEx

When a call to the **EcDoAsyncConnectEx** message 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.

3.1.4 Higher-Layer Triggered Events

None.

3.1.5 Message Processing Events and Sequencing Rules

3.1.5.1 Notifying Client about Pending Notifications

3.1.5.1.1 Sending RopPending

The server SHOULD send a **RopPending** response command to the client whenever there are pending notifications on the session context associated with the client and any linked session contexts.

3.1.5.1.2 Sending Push Notification Datagram

The server MUST NOT take any actions if the client has not previously registered a callback address by using the **EcRRegisterPushNotification** method.

The server MUST send a datagram to the callback address when a notification is available for the client. The datagram sent by the server MUST contain the opaque data that was provided by the client when the callback address was registered.

The server MUST continue sending a datagram to the callback address at periodic intervals if event details are still queued for the client. The server SHOULD stop sending datagrams only when all of the notifications have been retrieved from the server through **EcDoRpcExt2** calls.

3.1.5.1.3 Receiving and Completing Asynchronous RPC call

Whenever an asynchronous call to **EcDoAsyncWaitEx** on interface **AsyncEMSMDB** 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** call. 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 for a specified amount of time. If the server already has a call outstanding for the same session context handle, the server SHOULD complete the new call.

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 output field *pulFlagsOut*. The server MUST return 0 (zero) in the *pulFlagsOut* output parameter if the call was completed when there is no notification for the client session.

3.1.5.2 Sending Notification Details

3.1.5.2.1 Sending RopNotify

The server SHOULD send a **RopNotify** response command to the client whenever there are pending notifications on the session context that is associated with the client. The server SHOULD send as many notification details through multiple **RopNotify** response commands as the ROP response buffer allows. If the server was not able to fit the details for all pending notifications into the ROP response buffer, it SHOULD also send a **RopPending** response command if the response buffer allows.

3.1.6 Timer Events

None.

3.1.7 Other Local Events

None.

3.2 Client Details

3.2.1 Abstract Data Model

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

3.2.2 Timers

None.

3.2.3 Initialization

3.2.3.1 Subscribing for Notifications

3.2.3.1.1 Sending RopRegisterNotification

If the client needs to receive notifications from the server, the client SHOULD send a **RopRegisterNotification** message to the server. The client MUST provide specific details about notifications it needs to receive and the scope of the notifications, as specified in section [2.2.1.2.1](#). Upon receiving the 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.

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

3.2.3.1.2 Sending RopSynchronizationOpenAdvisor

If the client needs to receive *StatusObjectModified* notifications, it MUST first create an ICS Advisor object by sending a **RopSynchronizationOpenAdvisor** message. The client MUST save the returned handle to the ICS Advisor object. When the client no longer needs to receive *StatusObjectModified* notifications, the handle of the ICS Advisor object MUST be released by using the **RopRelease** ROP.

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

3.2.3.1.3 Sending RopRegisterSynchronizationNotifications

After the ICS Advisor object has been created by using **RopSynchronizationOpenAdvisor** message, the client SHOULD define the scope of notifications by using **RopRegisterSynchronizationNotifications**. The client can send **RopRegisterSynchronizationNotifications** multiple times to the server.

3.2.3.1.4 Sending RopSetSynchronizationNotificationGuid

If the client needs to suppress *StatusObjectModified* notifications on certain operations, it SHOULD assign a *StatusObjectModified* with a special GUID by means of **RopSetSynchronizationNotificationGuid**. If the client has assigned a GUID to the

StatusObjectModified, the client MUST set the value of the **PidTagChangeNotificationGuid** property to the Logon object to suppress *StatusObjectModified* notifications for the operations made by using that logon.

3.2.3.1.5 Subscribing for Table Notifications

The client MUST NOT take any actions to subscribe to table notifications. The subscription is created automatically when the client creates a **Table object** on the server.

3.2.3.2 Initializing Push Notifications

3.2.3.2.1 Sending EcRRegisterPushNotifications

The client calls **EcRRegisterPushNotification** to register a callback address for the session context. 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 3.1.4.11) by using the TCP/IP protocol, it registers an AF_INET callback address in the **EcRRegisterPushNotification** call.

Clients connecting by means of the RPC/HTTP protocol SHOULD NOT use the push notification method of being notified of pending event information. The client SHOULD use either the basic polling method or the asynchronous RPC notification method, as described in sections [1.3.1.2](#) and [1.3.1.4](#), respectively.

Because of network conditions such as firewalls or the use of RPC/HTTP connections by the client, it is not always possible for the 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 **EcRRegisterPushNotification**, until it receives a datagram from the server. When the client receives a 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 **EcDoRpcExt2** to retrieve event information.

3.2.3.2.2 Sending EcDoAsyncConnectEx

The client SHOULD determine whether the server supports **EcDoAsyncConnectEx** by examining the server version information that is returned from the **EcDoConnectEx** call, as specified in [\[MS-OXCRPC\]](#) section 3.1.4.11. For details about which minimum server version is required for using the asynchronous RPC notification method, see section [1.7](#).

The client can call **EcDoAsyncConnectEx** after a successful **EcDoConnectEx** call. The client MUST save the returned asynchronous context handle after the **EcDoAsyncConnectEx** call completes. The client MUST use the asynchronous context handle in the subsequent **EcDoAsyncWaitEx** calls to the server.

3.2.4 Higher-Layer Triggered Events

None.

3.2.5 Message Processing Events and Sequencing Rules

3.2.5.1 Receiving Notification About Pending Notifications

3.2.5.1.1 Receiving RopPending

Upon receiving **RopPending** in the response buffer of **EcDoRpcExt2**, the client MUST determine whether the session index provided in the **RopPending** matches any of the sessions created by the client. If the session index matches, the client SHOULD make **EcDoRpcExt2** calls 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** does not match the index of any session created by the client, the client MUST NOT take any actions.

3.2.5.1.2 Receiving Push Notification Datagram

Upon receiving a datagram on the callback address that was previously registered by the client by means of **EcRRegisterPushNotification**, the client MUST verify that the content of the 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 **EcRRegisterPushNotification**. If the content of the datagram is valid, the client SHOULD make **EcDoRpcExt2** calls to receive notification details from the server. Otherwise, the client MUST NOT take any actions on the datagram.

3.2.5.1.3 Sending and Receiving EcDoAsyncWaitEx

If the server supports asynchronous RPC notifications, and the client successfully created asynchronous context handle by calling **EcDoAsyncConnectEx**, the client SHOULD call **EcDoAsyncWaitEx** to determine whether notifications are pending on the server.

When a call to **EcDoAsyncWaitEx** completes, the client MUST examine its return value and the value of the *pulFlagsOut* output parameter. If the return value is "0x00000000" and bit "0x00000001" is set in the *pulFlagsOut* output parameter, the client SHOULD make **EcDoRpcExt2** calls to receive notification details from the server.

After the results of **EcDoAsyncWaitEx** are processed, the client SHOULD call **EcDoAsyncWaitEx** again to continue to listen for more notifications.

3.2.5.2 Receiving Notification Details

3.2.5.2.1 Receiving RopNotify

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

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                // NewMail
    80                // Message
    010000000078291F  // Message FID
    0100000000783484  // Message MID

    22000000          // MessageFlags
    00                // ASCII
    49504D2E4E6F746500 // Message class
  </Buffer>
</Data>

<Data name="ObjectCreatedNotification">
  <Buffer>
    04                // ObjectCreated
    00                // No flags
    0100000000782781  // Object FID
    0100000000782780  // Parent FID
    0000              // Number of PTAGs
  </Buffer>
</Data>

<Data name="ObjectCreatedNotification">
  <Buffer>
    04                // ObjectCreated
    80                // Message
    0100000000782780  // Message FID
    0100000000784172  // Message MID

    1F00              // Number of PTAGs
    0B001B0E          // PTAGs...
    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                // ObjectDeleted
    00                // No flags
    0100000000782780 // Folder FID
    010000000078277F // Parent FID
  </Buffer>
</Data>

<Data name="ObjectModifiedNotification">
  <Buffer>
    10                // ObjectModified
    00                // No flags
    0100000000782780 // Object FID

    0200                // Number of PTAGs
    03003866            // Ptags...
    0B000A36
  </Buffer>
</Data>

<Data name="ObjectModifiedNotification">
  <Buffer>
    10                // ObjectModified
    20                // UnreadItemsChanged
    010000000078291F // Object FID
    0100                // Number of PTAGs
    03000336            // Ptag
    00000000            // Value of unread items changes
  </Buffer>
</Data>

<Data name="ObjectModifiedNotification">
  <Buffer>
    10                // ObjectModified
    10                // TotalItemsChanged
    0100000000782780 // Object FID

    0400                // Number of PTAGs
    03000236            // Ptags...
    0300080E
    0300AF66
    0300B366
  </Buffer>
</Data>

```

```

        01000000          // Value of total items changes
    </Buffer>
</Data>

<Data name="ObjectModifiedNotification">
    <Buffer>
        10                // ObjectModified
        30                // UnreadItemsChanged
        010000000078291F // Object FID

        0500              // Number of PTAGs
        03000236           // Ptags...
        03000336
        0300080E
        0300AF66
        0300B366

        04000000          // Value of total items changes
        03000000          // Value of unread items changes
    </Buffer>
</Data>

<Data name="ObjectMovedNotification">
    <Buffer>
        20                // ObjectMoved
        80                // Message
        0100000000782781 // Message FID
        0100000000784378 // Message MID
        0100000000782780 // Old message FID
        0100000000784172 // Old message MID
    </Buffer>
</Data>

<Data name="ObjectCopiedNotification">
    <Buffer>
        40                // ObjectCopied
        80                // Message
        0100000000782780 // Message FID
        0100000000784173 // Message MID
        0100000000782780 // Old message FID
        0100000000784172 // Old message MID
    </Buffer>
</Data>

<Data name="TableModifiedNotification">
    <Buffer>
        00 01            // NotificationType = Hierarchy
        01 00            // TableModifiedNotificationType = TableChanged
    </Buffer>
</Data>

<Data name="TableModifiedNotification">
    <Buffer>
        00 01            // NotificationType = Hierarchy
        07 00            // TableModifiedNotificationType = TableRestrictDone
    </Buffer>
</Data>

<Data name="TableRowAddModifiedNotification">

```

```

<Buffer>
  00 01          // NotificationType (Hierarchy)
  03 00          // TableModifiedNotificationType (Added)
  01 00 00 02 81 6C EA 9D // FID
  01 00 00 02 81 6C EA 9E // InsertAfterFID

  A3 00 // Size of the property row

  // Values for the columns of the new row
  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 = Contents (TableModified | SearchFolder |
Message)
    03 00          // TableModifiedNotificationType (Added)
    01 00 00 00 00 78 60 45 // FID
    01 00 00 02 81 6C FC 84 // MID
    01 00 00 00 // Instance
    01 00 00 00 00 78 60 45 // InsertAfterFID
    01 00 00 02 81 6C FC 82 // InsertAfterMID
    01 00 00 00 // InsertAfterInstance

    A3 00 // Size of the property row

    // Values for the columns of the new row
    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 (Hierarchy)
    05 00          // TableModifiedNotificationType (Modified)
    01 00 00 00 00 78 60 45 // FID
    01 00 00 00 00 78 60 50 // InsertAfterFID

    A3 00 // Size of the property row

    // Values for the columns of the new row
    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 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 (Contents)
    05 00          // TableModifiedNotificationType (Modified)
    01 00 00 00 00 78 60 45 // FID
    01 00 00 02 81 6C FC 83 // MID
    01 00 00 00 // Instance
    01 00 00 00 00 78 60 46 // InsertAfterFID
    01 00 00 02 81 6C FC 84 // InsertAfterMID
    01 00 00 00 // Insert after instance

    A3 00 // Size of the property row

    // Values for the columns of the new row
    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 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 = Hierarchy (TableModified)
    04 00      // TableModifiedNotificationType = Deleted
    01 00 00 00 00 78 60 45 // FID
  </Buffer>
</Data>

<Data name="TableRowDeletedModifiedNotification">
  <Buffer>
    00 C1      // NotificationType = Contents (TableModified | SearchFolder |
Message)
    04 00      // TableModifiedNotificationType = Deleted
    01 00 00 02 81 6C EA 96 // FID
    01 00 00 02 81 6D 09 01 // MID
    01 00 00 00 // Instance
  </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 specified in [\[MS-OXCROPS\]](#) do apply to this protocol.

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® Office Outlook® 2003
- Microsoft® Office Outlook® 2007
- Microsoft® Outlook® 2010

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 1.3.1.3:](#) Office Outlook 2007 and Outlook 2010 use either the basic polling method or the asynchronous RPC notification method described in section [1.3.1.4](#).

[<2> Section 1.7:](#) Exchange 2007, Exchange 2010, Office Outlook 2007, and Outlook 2010 support asynchronous RPC notifications.

[<3> Section 2.2.1.1:](#) Exchange 2003, Exchange 2007, and Exchange 2010 cannot trigger this event.

[<4> Section 2.2.1.2.1.1:](#) Exchange 2010 returns "NotImplemented" for a **RopRegisterNotification** request with a **CriticalError** notification type.

[<5> Section 2.2.1.2.2:](#) Exchange 2010 does not implement the creation of this object.

[<6> Section 2.2.1.2.2:](#) Exchange 2010 parses these ROPs, but the return value is "Not Implemented".

[<7> Section 2.2.1.4.1:](#) The server returns ANSI values for Office Outlook 2003 and Office Outlook 2007 clients if the client is running in "cached mode".

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

A

Abstract data model
 [client](#) 22
 [server](#) 19
[Applicability](#) 8

C

[Capability negotiation](#) 8
[Change tracking](#) 33
Client
 [abstract data model](#) 22
 [higher-layer triggered events](#) 23
 [other local events](#) 24
 [timer events](#) 24
 [timers](#) 22

D

Data model - abstract
 [client](#) 22
 [server](#) 19

F

[Fields - vendor-extensible](#) 8

G

[Glossary](#) 5

H

Higher-layer triggered events
 [client](#) 23
 [server](#) 20

I

[Implementer - security considerations](#) 31
[Informative references](#) 6
[Introduction](#) 5

M

Messages
 [transport](#) 9

N

[Normative references](#) 6

O

Other local events
 [client](#) 24
 [server](#) 22
[Overview \(synopsis\)](#) 6

P

[Preconditions](#) 8
[Prerequisites](#) 8
[Product behavior](#) 32

R

References
 [informative](#) 6
 [normative](#) 6
[Relationship to other protocols](#) 7

S

Security
 [implementer considerations](#) 31
Server
 [abstract data model](#) 19
 [higher-layer triggered events](#) 20
 [other local events](#) 22
 [timer events](#) 21
 [timers](#) 19
[Standards assignments](#) 8

T

Timer events
 [client](#) 24
 [server](#) 21
Timers
 [client](#) 22
 [server](#) 19
[Tracking changes](#) 33
[Transport](#) 9
Triggered events - higher-layer
 [client](#) 23
 [server](#) 20

V

[Vendor-extensible fields](#) 8
[Versioning](#) 8