[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.mspx). 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.
- **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.
10/07/2011	7.0	Major	Significantly changed 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 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
handle
Hypertext Transfer Protocol (HTTP)
remote procedure call (RPC)
Unicode

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

asynchronous context handle
remote operation (ROP)
ROP request buffer
ROP response buffer
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-OXCDATA] Microsoft Corporation, "Data Structures".

[MS-OXCFOLD] 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".

[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, "Configuring a Static UDP Port for Push Notifications in an Exchange 2010 Environment", http://social.technet.microsoft.com/wiki/contents/articles/configuring-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".

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 2.2.14.2).

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 requires 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 2.2.14.3**)** 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 is required 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 [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 remote procedure call (RPC) transport, as specified in [MS-OXCRPC].

Internet 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

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.
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 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 that specifies the nature of the change (content or hierarchy), the FID or MID value, and new table values. The **TableRowAdded**, **TableRowDeleted**, **TableRowModified**, and **TableRestrictionChanged** events each produce informative notifications.
- Generates a basic notification that does not include specifics about the change made. The **TableChanged** event produces a basic notification.
- 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 informative notifications whenever possible and only generate a basic notification when an informative notification is not feasible.

For more details about subscribing to **TableModified** event notifications, see section 3.1.3.1.2.

2.2.1.2 Subscription Management

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 following table describes the parts of the **notification subscription** request.

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

2.2.1.2.1.1 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
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.

Value	Meaning
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 "0" (zero). The server behavior is undefined if the value is not 0.

WantWholeStore (1 byte): "TRUE" (nonzero) if the scope for notifications is the entire database; otherwise, "FALSE" (zero).

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 2.2.14.3.

2.2.1.3.2 EcRRegisterPushNotification

EcRRegisterPushNotification<5> 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 ([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.

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].

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, an **ICS Advisor object**, 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	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.
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): 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. If this bit is set, bit "0x800"0 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.

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

TableRowFolderID (8 bytes): The folder ID (FID), 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 message ID (MID), 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 FID of the item triggering the notification. This field is available onlyl if the **TableEventType** field is available and is "0x0003" or "0x0005".

InsertAfterTableRowID (8 bytes): The old MID 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".

HierarchyChanged (1 byte): This field is available only if the **NotificationType** value in the **NotificationFlags** field is "0x0200". If set, the value of this field is **TRUE** (0x01).

FolderIDNumber (4 bytes): The number FIDs in the **FolderIDs** field. This field is available only if the **NotificationType** value in the **NotificationFlags** field is "0x0200".

FolderIDs (variable): An array of FIDs. This field is available only if the **NotificationType** value in the **NotificationFlags** field is "0x0200".

ICSChangeNumbers (variable): An array of older 32-bit change numbers. This field is available only if the **NotificationType** value in the **NotificationFlags** field is "0x0200".

FolderId (8 bytes): The FID 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".

MessageId (8 bytes): The MID 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 (8 bytes): The FID 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 (8 bytes): The old FID 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 MID 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 (8 bytes): The old parent FID 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 (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 **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 (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 **TagCount** value is not "0x00" 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 "0x100" 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 **NotificationType** value in the **NotificationFlags** field is "0x0002". For details, see [MS-OXCMSG] section 2.2.1.6.

UnicodeFlag (1 byte): 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**. This field is available only if the value of the **NotificationType** field in the **NotificationFlags** field is "0x0002".<6>

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" (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".

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 ADM type:

NotificationObject: An object that receives and tracks notifications.

3.1.2 Timers

If push notifications are supported by the server, as specified in section 3.1.5.1.2, the server SHOULD allow for a one second 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 Subscribing for Table Notifications

The server SHOULD<7> require that a table view is created in order to send **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, see [MS-OXCFOLD].

The server SHOULD<8> 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**, or **RopSeekRowBookmark**.

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 SHOULD<9> support sending push notification datagrams.

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

If the server supports sending push notification datagrams, 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.

If the server supports sending push notification datagrams, the server MUST continue sending a datagram to the callback address at one second 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 5 minutes. If the server already has a call outstanding for the same session context handle, the server SHOULD complete the new call and return ecRejected, as specified in [MS-OXCRPC] section 3.1.4.15, 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 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** ROP (section 2.2.1.4.1) 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 **RopNotify** responses 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** ROP 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

None.

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 Subscribing for Table Notifications

The client SHOULD<10> send the one of the following ROPs to the server, which will cause 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.

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

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
```

```
        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       // TableModifiedNotificationTypeType = TableChanged
    </Buffer>
</Data>

<Data name="TableModifiedNotification">
    <Buffer>
        00 01       // NotificationType = Hierarchy
        07 00       // TableModifiedNotificationTypeType = 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 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 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.

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® 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 2.2.1.3.2: Exchange 2003 and Exchange 2007 support the **EcRRegisterPushNotification** RPC ([MS-OXCRPC] section 3.1.4.5). The initial release version of Exchange 2010 does not support the **EcRRegisterPushNotification** RPC, and the returned value will always be **ecNotSupported**. Exchange 2010 SP2 does support the **EcRRegisterPushNotification** RPC if a registry key is created to support push notifications, as described in [MSFT-ConfigStaticUDPPort].

<6> Section 2.2.1.4.1.1: The server returns ANSI values for Office Outlook 2003 and Office Outlook 2007 clients if the client is working in cached mode, as specified by the **ClientMode** field in [MS-OXCRPC] section 2.2.2.6.

<7> Section 3.1.3.1.2: Exchange 2003 and Exchange 2007 do not require that a table view is created in order to send table notifications.

<8> Section 3.1.3.1.2: Exchange 2003 and Exchange 2007 do not stop sending notifications if the **RopResetTable** ROP ([MS-OXCROPS] section 2.2.5.15) is received.

<9> Section 3.1.5.1.2: Exchange 2003 and Exchange 2007 support push notifications. The initial release version of Exchange 2010 does not support push notifications. Exchange 2010 SP2 does support push notifications if a registry key is created, as described in [MSFT-ConfigStaticUDPPort].

<10> Section 3.2.3.1.2: 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.

7 Change Tracking

This section identifies changes that were made to the [MS-OXCNOTIF] protocol document between the August 2011 and October 2011 releases. Changes are classified as New, Major, Minor, Editorial, or No change.

The revision class **New** means that a new document is being released.

The revision class **Major** means that the technical content in the document was significantly revised. Major changes affect protocol interoperability or implementation. Examples of major changes are:

- A document revision that incorporates changes to interoperability requirements or functionality.
- An extensive rewrite, addition, or deletion of major portions of content.
- The removal of a document from the documentation set.
- Changes made for template compliance.

The revision class **Minor** means that the meaning of the technical content was clarified. Minor changes do not affect protocol interoperability or implementation. Examples of minor changes are updates to clarify ambiguity at the sentence, paragraph, or table level.

The revision class **Editorial** means that the language and formatting in the technical content was changed. Editorial changes apply to grammatical, formatting, and style issues.

The revision class **No change** means that no new technical or language changes were introduced. The technical content of the document is identical to the last released version, but minor editorial and formatting changes, as well as updates to the header and footer information, and to the revision summary, may have been made.

Major and minor changes can be described further using the following change types:

- New content added.
- Content updated.
- Content removed.
- New product behavior note added.
- Product behavior note updated.
- Product behavior note removed.
- New protocol syntax added.
- Protocol syntax updated.
- Protocol syntax removed.
- New content added due to protocol revision.
- Content updated due to protocol revision.
- Content removed due to protocol revision.
- New protocol syntax added due to protocol revision.

- Protocol syntax updated due to protocol revision.
- Protocol syntax removed due to protocol revision.
- New content added for template compliance.
- Content updated for template compliance.
- Content removed for template compliance.
- Obsolete document removed.

Editorial changes are always classified with the change type **Editorially updated.**

Some important terms used in the change type descriptions are defined as follows:

- **Protocol syntax** refers to data elements (such as packets, structures, enumerations, and methods) as well as interfaces.
- **Protocol revision** refers to changes made to a protocol that affect the bits that are sent over the wire.

The changes made to this document are listed in the following table. For more information, please contact protocol@microsoft.com.

Section	Tracking number (if applicable) and description	Major change (Y or N)	Change type
2.1 Transport	Added a reference to [RFC768].	N	Content updated.
2.2.1.1.1 TableModified Event Types	Added details about informative and basic notifications.	N	Content updated.
2.2.1.1.1 TableModified Event Types	Removed the TableError, TableSortDone, and TableColumnsChanged table events as they are never sent by the server.	Y	Content removed.
2.2.1.2.1.1 RopRegisterNotification ROP Request Buffer	Added a section that specifies the RopRegisterNotification ROP request buffer.	N	Content updated.
2.2.1.3.2 EcRRegisterPushNotification	Added a product behavior note that specifies the support for the EcRRegisterPushNotification RPC.	Y	New product behavior note added.
2.2.1.4.1 RopNotify	Moved the description of the ROP fields to the "RopNotify ROP Response Buffer" section.	N	Content updated.
2.2.1.4.1.1 RopNotify ROP Response Buffer	Removed the FALSE value description from the HierarchyChanged field description as the value is always TRUE, if set.	N	Content updated.
2.2.1.4.1.1 RopNotify ROP Response Buffer	Added a section to specify the response buffer of the RopNotify ROP.	N	Content updated.
2.2.1.4.1.1	Updated the definition of the TableRowData	N	Content

Section	Tracking number (if applicable) and description	Major change (Y or N)	Change type
RopNotify ROP Response Buffer	field.		updated.
2.2.1.4.1.1 RopNotify ROP Response Buffer	Changed the TableEventType values to two byte values.	N	Content updated.
3.1.1 Abstract Data Model	Added details about the NotificationObject ADM type.	N	Content updated.
3.1.3.1.2 Subscribing for Table Notifications	Added product behaviosr note for the server's behavior related to sending TableModified event notifications.	Y	New product behavior note added.
3.1.3.1.2 Subscribing for Table Notifications	Added a requirement that a table view is created before a TableModified event notification is sent to the client.	Y	Content updated.
3.1.5.1.3 Receiving and Completing Asynchronous RPC call	Added details that the server should not complete the call until the call has been outstanding for 5 minutes.	N	Content updated.
3.1.5.1.3 Receiving and Completing Asynchronous RPC call	Added details about the server behavior when the server already has a call outstanding for the same session context handle.	N	Content updated.
3.1.5.2.1 Sending RopNotify	Clarified details about adding RopNotify response buffers to the EcDoRpcExt method response buffer.	N	Content updated.
3.2.1 Abstract Data Model	Updated the content to "None."	N	Content updated.
3.2.3.1.2 Subscribing for Table Notifications	Added a requirement that the client sends RopQueryRows and RopFindRow before a TableModified event notification is sent to the client.	Y	Content updated.
3.2.3.1.2 Subscribing for Table Notifications	Added product behavior note for the client's behavior related to subscribing to TableModified event notifications.	Y	New product behavior note added.
5.2 Index of Security Parameters	Added section.	N	New content added for template compliance.
	Removed the "NotificationTypes" section. Moved the content into the RopRegisterNotification ROP Request Buffer section.	N	Content updated.
	Removed the "RopSynchronizationOpenAdvisor", "RopRegisterSynchronizationNotifications", and "RopSetSynchronizationNotificationGuid" topics	Y	Content removed.

Section	Tracking number (if applicable) and description	Major change (Y or N)	Change type
	from the specification.		
	Removed the "Receiving RopSynchronizationOpenAdvisor", "Receiving RopRegisterSynchronizationNotifications", and "Receiving RopSetSynchronizationNotificationGuid" sections from the specification.	Y	Content removed.
	Removed the "Sending RopSynchronizationOpenAdvisor", "Sending RopRegisterSynchronizationNotifications", and "Sending RopSetSynchronizationNotificationGuid" sections from the specification.	Y	Content removed.
	Removed the "NotificationFlags" section. Moved the content into the RopNotify ROP Response Buffer section.	N	Content removed.
	Removed the "TableEventType" section. Moved the content into the RopNotify ROP Response Buffer section.	N	Content removed.
	Removed the "MessageFlags" section. Moved the content into the RopNotify ROP Response Buffer section.	N	Content removed.
	Removed the "MessageClass" section. Moved the content into the RopNotify ROP Response Buffer section.	N	Content removed.

8 Index