[MS-OXCNOTIF]: Core Notifications Protocol Specification

Intellectual Property Rights Notice for Protocol Documentation

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

Revision Summary			
Author	Date	Version	Comments
Microsoft Corporation	April 4, 2008	0.1	Initial Availability.
Microsoft Corporation	June 27, 2008	1.0	Initial Release.

Table of Contents

1 Introduction

This document specifies a protocol for transmitting notifications to a client about certain events on a server. This protocol is commonly used to inform the client about changes that occurred in folders and messages on the server.

1.1 Glossary

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

> **ASCII**
> **Asynchronous Context Handle (AXCH)**
> **binary large object (BLOB)**
> **change number (CN)**
> **folder ID**
> **GUID**
> **handle**
> **Logon object**
> **message ID**
> **remote operation (ROP)**
> **remote procedure call (RPC)**
> **ROP request buffer**
> **ROP response buffer**
> **Session Context Handle (CXH)**
> **Unicode**

The following terms are defined in this document:

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

Internet datagram: The unit of data exchanged between a pair of Internet modules (includes the Internet header).

notification: A message the client receives when a specific event occurs on the server.

notification subscription: A request to receive **notifications** from the server.

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

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

1.2 References

1.2.1 Normative References

[MS-OXCFXICS] Microsoft Corporation, "Bulk Data Transfer Protocol Specification", April 2008.

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

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

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

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

[MS-OXCTABL] Microsoft Corporation, "Table Object Protocol Specification", April 2008.

[MS-OXGLOS] Microsoft Corporation, "Office Exchange Protocols Master Glossary", April 2008.

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

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.

1.3 Protocol Overview

The messaging client can register to receive **notifications** about certain events that can happen 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], in the format described by **RopNotify**, as specified in [MS-OXCROPS].

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 the two parts of the protocol.

1.3.1   Pending Notifications

Because the receipt of **notification** details is only done through the **ROP response buffer** that is returned from **EcDoRpcExt2** 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 **EcDoRpcExt2**. 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 **RopPending**, as specified in [MS-OXCROPS], in the response buffer on **EcDoRpcExt2** call.

1.3.1.2   Polling

If a client is idle and is not making **EcDoRpcExt2** calls, it cannot receive **RopNotify**. The simplest way for a client to retrieve **notification** details is to make **EcDoRpcExt2** calls on regular intervals. The server allows the client to call **EcDoRpcExt2** with no **remote operation (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** calls. The output parameter *pcmsPollsMax* in both of these calls contains the number of milliseconds 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, it may not use the polling method.

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 via RPC/HTTP protocol may use the Push Notification method of being signaled of pending notifications. The client either uses the basic polling method or the Asynchronous RPC Notification method described in section 1.3.1.4.

1.3.1.4   Asynchronous RPC Notification

Asynchronous RPC Notification method allows the client to make an asynchronous RPC call to the server where 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 where the Push Notification method will not. The client determines if the server supports this notification method by examining the server version information that is returned from the

EcDoConnectEx call. See section 0 to determine which minimum server version is required to utilize the Asynchronous RPC Notification method.

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 through 1.3.1.4, the client calls **EcDoRpcExt2** 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 was 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

The Core Notifications protocol specification provides a low-level explanation of notifying a client about events on the server. For information about the application of this protocol in a MAPI provider, see [MSDN-ENM].

This specification relies on an understanding of [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:

- Notifying clients about certain events on the server.

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

This protocol provides basic information, high efficiency, and complete preservation of data fidelity for these uses. It might not be appropriate for use in scenarios that do 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 from or to a mailbox.

1.7   Versioning and Capability Negotiation

This document 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 RPC Notifications to be implemented. The client examines the server version to determine if Asynchronous RPC Notifications are supported. See [MS-OXCRPC] for more details about how to determine server version. <1>

- **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 respectively using the underlying **ROP request buffers** and **ROP response buffers**, as specified in [MS-OXCROPS].

Asynchronous calls are made on the server using **remote procedure call (RPC)** transport, as specified in [MS-OXCRPC].

Datagrams are sent from server to client 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 MAY register to receive notifications about these events.

Table 1: Server Events

Name	Description
CriticalError	A critical error has occurred on the server.
NewMail	A new e-mail message has been received by the server.
ObjectCreated	A new item has been created on the server
ObjectDeleted	An existing item has been deleted from the server.
ObjectModified	An existing item has been modified on the server.
ObjectMoved	An existing item has been moved to another location on the server.
ObjectCopied	An existing item has been copied on the server.
SearchComplete	A search operation has been completed on the server.
TableModified	A table was modified on the server. For details about this event, see section 2.2.1.1.1.
StatusObjectModified	An ICS state has been modified on the server.

2.2.1.1.1 TableModified Event Types

The following table describes the table modification event types.

Table 2:   Table Modification Events

Name	Description
TableChanged	A table has been changed.
TableError	An error 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.
TableSortDone	A table sort has been completed.
TableRestrictionChanged	A table restriction has been changed.
TableColumnsChanged	Table columns have been changed.
TableReload	A table has been reloaded.

2.2.1.2 Subscription Management

2.2.1.2.1 RopRegisterNotification

RopRegisterNotification creates a subscription for specified **notifications** on the server and returns a **handle** of the subscription to the client.

Table 3:   Notification Subscription Request

Name	Type	Size	Description
InputHandle	Handle	4	Handle of the **Logon object**. See [MS-OXCROPS] for more details.
NotificationTypes	Byte	1	A set of bits describing notifications that the client is interested in receiving.

Name	Type	Size	Description
			See 2.2.1.2.1.1.
Reserved	Byte	1	The field is reserved. The field value MUST be zero. The behavior is undefined if the value is not zero.
EntireDatabase	Byte	1	Set to **TRUE** (non-zero) if the scope for notifications is the entire database. Set to **FALSE** (zero) otherwise.
ScopeFolderID	ID	8	ID of the folder to limit the scope of notifications. This field is available only if **EntireDatabase** is zero.
ScopeMessageID	ID	8	ID of the message inside the folder referenced by **ScopeFolderID** to limit the scope for notifications. This field is available only if **EntireDatabase** is zero.

Table 4: Notification Subscription Response

Name	Type	Size	Description
OutputHandle	Handle	4	**Handl**e of the Notification Subscription object created by this **ROP**. See [MS-OXCROPS] for more details.

2.2.1.2.1.1   NotificationTypes

The following table lists the notification types that are available.

Value	Meaning
0x01	The server MUST send **notifications** to the client when *CriticalError* events occur within the scope of interest.
0x02	The server MUST send notifications to the client when *NewMail* events occur within the scope of interest.
0x04	The server MUST send notifications to the client when *ObjectCreated* events occur within the scope of interest.
0x08	The server MUST send notifications to the client when *ObjectDeleted* events occur within the scope of interest.
0x10	The server MUST send notifications to the client when *ObjectModified* events occur within the scope of interest.
0x20	The server MUST send notifications to the client when *ObjectMoved* events occur within the scope of interest.
0x40	The server MUST send notifications to the client when *ObjectCopied* events occur within the scope of interest.
0x80	The server MUST send notifications to the client when *SearchCompleted* events occur within the scope of interest.

See section 2.2.1.1 for details about server events.

2.2.1.2.2 RopSynchronizationOpenAdvisor

RopSynchronizationOpenAdvisor creates an ICS Advisor object on the server and returns a handle of the object to the client.

Table 5: ICS Advisor Request

Name	Type	Size	Description
InputHandle	Handle	4	**Handle** of the Logon object. See [MS-OXCROPS] for more details.

Table 6: ICS Advisor Response

Name	Type	Size	Description
OutputHandle	Handle	4	**Handle** of the ICS Advisor object created by this **ROP**. See [MS-OXCROPS] for more details.

2.2.1.2.3 RopRegisterSynchronizationNotifications

RopRegisterSynchronizationNotifications creates a subscription for *StatusObjectModified* notifications on the server.

Table 7: StatusObjectModified Notification Request

Name	Type	Size	Description
InputHandle	Handle	4	Handle of the ICS Advisor object.
NumberOfFolderIDs	Short	2	Number of **folder ID**s that limit the scope of the **notification subscription**.
FolderIDs	ID[]	*Number OfFolder IDs*	List of folder IDs that limit the scope of the notification subscription.
ChangeNumbers	ULong []	*Number OfFolder IDs*	List of folder **change numbers (CNs)**.

For details about the response, see [MS-OXCROPS].

2.2.1.2.4 RopSetSynchronizationNotificationGuid

RopSetSynchronizationNotificationGuid assigns a notification **GUID** to an ICS Advisor object on the server.

Table 8: Notification GUID Request

Name	Type	Size	Description
InputHandle	Handle	4	**Handle** of the ICS Advisor object. See [MS-OXCROPS] for more details.
NotificationGuid	GUID	16	A notification GUID to assign to the ICS Advisor object.

For details about the response, see [MS-OXCROPS].

2.2.1.3 Pending Notifications

2.2.1.3.1 RopPending

RopPending notifies the client that there are pending **notifications** on the server for the client. This **ROP** MUST appear only in response buffers of **EcDoRpcExt2**. See [MS-OXCROPS] for more details.

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] for more details.

2.2.1.3.3 EcDoAsyncConnectEx

EcDoAsyncConnectEx is an RPC method that is used to acquire an **Asynchronous Context Handle (ACXH)** on the server to use in subsequent **EcDoAsyncWaitEx** calls. See [MS-OXCRPC] for more details.

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. See [MS-OXCRPC] for more details.

2.2.1.4 Notification Details

2.2.1.4.1 RopNotify

RopNotify provides the client with the details of **notifications** that are sent by server. This **ROP** MUST appear only in response buffers of **EcDoRpcExt2**.

Table 9: RopNotify Response

Name	Type	Size	Description
NotificationHandle	Handle	4	Handle of the target object for the notification. The target object can be a notification subscription, an ICS Advisor, or a table.
NotificationFlags	Short	2	Set of bits describing the type of the notification and availability of the notification data fields. See section 2.2.1.4.1.1.
TableEventType	Byte	1	Subtype of the notification for a *TableModified* event. This field is available only if the **NotificationType** value in **NotificationFlags** is 0x0100.

Name	Type	Size	Description
			See section 2.2.1.4.1.2.
TableRowFolderID	ID	8	Folder ID of the item that is triggering this notification. This field is only available if the **TableEventType** field is available and is equal to 0x03, 0x04, or 0x05.
TableRowMessageID	ID	8	**Message ID** of the item triggering this notification. This field is only available if bit 0x8000 is set in **NotificationFlags** and **TableEventType** is available and is equal to 0x03, 0x04, or 0x05.
TableRowPreviousInstance	ULong	4	An identifier of the instance of the previous row in the table. See [MS-OXCTABL] for more details. This field is only available if bit 0x8000 is set in **NotificationFlags** and **TableEventType** is available and is equal to 0x03, 0x04, or 0x05.
TableRowOldFolderID	ID	8	Old folder ID of the item triggering this notification. This field is only available if the **TableEventType** field is available and is equal to 0x03 or 0x05.
TableRowOldMessageID	ID	8	Old message ID of the item triggering this notification. This field is only available if bit 0x8000 is set in **NotificationFlags** and **TableEventType** is available and is equal to 0x03 or 0x05.
TableRowDataSize	Short	2	Length of table row data. This field is only available if bit 0x8000 is set in **NotificationFlags** and **TableEventType** is available and is equal to 0x03 or 0x05.
TableRowData	String	*TableRow DataSize*	Table row data. This field is only available if bit 0x8000 is set in **NotificationFlags** and **TableEventType** is available and is equal to 0x03 or 0x05.
HierarchyChanged	Byte	1	Set to **TRUE** (non-zero) if folder hierarchy has changed. Set to **FALSE** (zero) otherwise. This field is available only if the **NotificationType** value in **NotificationFlags** is 0x0200.
FolderIDNumber	ULong	4	Number of folder IDs. This field is available only if the **NotificationType** value in **NotificationFlags** is 0x0200.

Name	Type	Size	Description
FolderIDs	GID[]	*FolderID Number*	Folder IDs. This field is available only if the **NotificationType** value in **NotificationFlags** is 0x0200.
ICSChangeNumbers	ULong[]	*FolderID Number*	Folder CNs. This field is available only if the **NotificationType** value in **NotificationFlags** is 0x0200.
FolderId	ID	8	Folder ID of the item triggering the event. This field is available only if the **NotificationType** value in **NotificationFlags** is not 0x0100, 0x0200, or 0x0400.
MessageId	ID	8	Message ID of the item triggering the event. This field is available only if the **NotificationType** value in **NotificationFlags** is not 0x0100, 0x0200, or 0x0400, and bit 0x8000 is set in **NotificationFlags**.
ParentFolderId	ID	8	Folder ID of the parent folder of the item triggering the event. This field is available only if the **NotificationType** value in **NotificationFlags** is 0x0004, 0x0008, 0x0020, or 0x0040 and bit 0x4000 is set or bit 0x8000 is not set in **NotificationFlags**.
OldFolderId	ID	8	Old folder ID of the item triggering the event. This field is available only if the **NotificationType** value in **NotificationFlags** is 0x0020 or 0x0040.
OldMessageId	ID	8	Old message ID of the item triggering the event. This field is available only if the **NotificationType** value in **NotificationFlags** is 0x0020 or 0x0040 and bit 0x8000 is set in **NotificationFlags**.
OldParentFolderId	ID	8	Old parent folder ID of the item triggering the event. This field is available only if the **NotificationType** value in **NotificationFlags** is 0x0020 or 0x0040 and bit 0x8000 is not set in **NotificationFlags**.
TagCount	Short	2	Number of property tags. This field is available only if the **NotificationType** value in **NotificationFlags** is 0x0004 or

Name	Type	Size	Description
			0x0010.
Tags	ULong[]	*TagCount*	List of IDs of properties that have changed. This field is available only if **TagCount** is available and **TagCount** is not equal to 0xFFFF.
TotalMessageCount	ULong	4	Total number of items in a folder triggering this event. This field is available only if bit 0x1000 is set in **NotificationFlags**.
UnreadMessageCount	ULong	4	Number of unread items in a folder triggering this event. This field is available only if bit 0x2000 is set in **NotificationFlags**.
MessageFlags	ULong	4	Message flags of new mail that has been received. This field is available only if the **NotificationType** value in **NotificationFlags** is 0x0002.
UnicodeFlag	Byte	1	Set to **TRUE** (non-zero) if **MessageClass** is in UNICODE. Set to **FALSE** (zero) otherwise. This field is available only if the **NotificationType** value in **NotificationFlags** is 0x0002.
MessageClass	String	*Variable*	Null-terminated string containing the message class of the new mail. The string is in **Unicode** if **UnicodeFlag** is **TRUE** (non-zero). The string is in **ASCII** if **UnicodeFlag** is **FALSE** (zero). This field is available only if the **NotificationType** value in **NotificationFlags** 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.

NotificationType is a 12 bit enumeration defining the type of the **notification**. The possible values are listed in the following table.

Value	Meaning
0x0001	The notification is for *CriticalError* event.

Value	Meaning
0x0002	The notification is for *NewMail* events.
0x0004	The notification is for *ObjectCreated* event.
0x0008	The notification is for *ObjectDeleted* event.
0x0010	The notification is for *ObjectModified* event.
0x0020	The notification is for *ObjectMoved* event.
0x0040	The notification is for *ObjectCopied* event.
0x0080	The notification is for *SearchCompleted* event.
0x0100	The notification is for *TableModified* events.
0x0200	The notification is for *StatusObjectModified* event.
0x0400	The value is reserved and MUST NOT be used.

The meaning of other flags is provided in the following table.

Value	Meaning
0x1000	**T** bit. The notification contains information about a change in total number of messages in a folder triggering the event. If this bit is set, then **NotificationType** MUST be 0x0010.
0x2000	**U** bit. The notification contains information about a change in number of unread messages in a folder triggering the event. If this bit is set, then **NotificationType** MUST be 0x0010.
0x4000	**S** bit. The notification is caused by an event in a search folder. If this bit is set, then bit 0x8000 MUST be set.
0x8000	**M** bit. The notification is caused by an event on a message.

2.2.1.4.1.2   TableEventType

Value	Meaning	
0x01		The notification is for *TableChanged* events.
0x02		The notification is for *TableError* events.
0x03		The notification is for *TableRowAdded* event.
0x04		The notification is for *TableRowDeleted* events.
0x05		The notification is for *TableRowModified* event.
0x06		The notification is for *TableSortDone* event.
0x07		The notification is for *TableRestrictionChanged* event.
0x08		The notification is for *TableColumnsChanged* event.
0x09		The notification is for *TableReloaded* event.

2.2.1.4.1.3   MessageFlags
See [MS-OXCMSG] for details.

2.2.1.4.1.4   MessageClass
See [MS-OXCMSG] for details.

3   Protocol Details

3.1   Notifications 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

None.

3.1.3   Initialization

3.1.3.1   Subscribing for Notifications

3.1.3.1.1   Receiving RopRegisterNotification

When a **RopRegisterNotification** 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 various fields of the **RopRegisterNotification** for future use.

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

3.1.3.1.2   Receiving RopSynchronizationOpenAdvisor

When **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 Advisors 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, *InputHandle* 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* 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 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 **RopSetSynchronizationNotificationGuid**. See [MS-OXCSTOR] for more details.

3.1.3.1.5 Subscribing for Table Notifications

The server SHOULD NOT require any special actions to register for **notifications** on table events. The server SHOULD create a subscription to table notifications 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.

3.1.3.2 Initializing Pending Notifications

3.1.3.2.1 Receiving EcRRegisterPushNotification

When a call to **EcRRegisterPushNotification** 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. See [MSDN-WS2] for more information.

The server SHOULD support a variety of different callback address types. The server SHOULD support at minimum the AF_INET address type for IP support and 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 immediately send a datagram containing the client's opaque data.

3.1.3.2.2 Receiving EcDoAsyncConnectEx

When a call to **EcDoAsyncConnectEx** is received by the server, the server MUST create an **ACXH** and MUST bind it to the **Session Context Handle (CXH)** used to make the call.

3.1.4   Message Processing Events and Sequencing Rules

3.1.4.1 Notifying Client about Pending Notifications

3.1.4.1.1 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.

3.1.4.1.2 Sending Push Notification Datagram

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

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 only stop sending datagrams when all the notifications have been retrieved from the server through **EcDoRpcExt2** calls. The server SHOULD allow for a certain time interval between datagrams until the client has retrieved all event information for the session. The server MAY provide server administrators a means to configure the time interval between the datagrams.

3.1.4.1.3 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 **ACXH** provided is a valid ACXH that was returned from **EcDoAsyncConnectEx**. 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 certain time. If the server already has a call outstanding for the same CXH, the server SHOULD immediately 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 zero in *pulFlagsOut* if the call was completed for any other reasons.

3.1.4.2 Sending Notification Details

3.1.4.2.1 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.

3.1.5 Timer Events

None.

3.1.6 Other Local Events

None.

3.2 Notifications 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** to the server. The client MUST provide specific details about notifications it needs to receive and the scope of the notification 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 **RopRelease**.

The client MAY send **RopRegisterNotification** 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 **RopSynchronizationOpenAdvisor**. 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 **RopRelease**.

The client MAY send **RopSynchronizationOpenAdvisor** multiple times to the sever.

3.2.3.1.3 Sending RopRegisterSynchronizationNotifications

After the ICS Advisor object has been created by using **RopSynchronizationOpenAdvisor**, the client SHOULD define the scope of **notifications** by using **RopRegisterSynchronizationNotifications**. The client MAY 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 an ICS Advisor object with a special **GUID** via **RopSetSynchronizationNotificationGuid**. If the client has assigned a GUID to the ICS Advisor object, 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 MAY register a variety of different callback address types if the server supports the address type. It is not required, but recommended that a client register a callback address 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** by using the TCP/IP protocol, it SHOULD register an AF_INET callback address in call **EcRRegisterPushNotification**.

Clients connecting via RPC/HTTP protocol SHOULD NOT use the Push Notification method of being signaled of pending event information. The client SHOULD either use the basic Polling method or the Asynchronous RPC Notification method described in sections 1.3.1.2 and 1.3.1.4.

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**, up until it receives a datagram from the server. When the client receives a datagram from the server at the 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. See section 1.7 to determine which minimum server version is required to utilize the Asynchronous RPC Notification method.

The client MAY call **EcDoAsyncConnectEx** after a successful **EcDoConnectEx** call. The client MUST save the returned ACXH after the **EcDoAsyncConnectEx** call completes. The client MUST use the **ACXH** in the subsequent **EcDoAsyncWaitEx** calls to the server.

3.2.4 Message Processing Events and Sequencing Rules

3.2.4.1 Receiving Notification About Pending Notifications

3.2.4.1.1 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 sever by using the **CXH** 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.

3.2.4.1.2 Receiving Push Notification Datagram

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

3.2.4.1.3 Sending and Receiving EcDoAsyncWaitEx

If the server supports Asynchronous RPC Notifications, and the client successfully created **ACXH** 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 sever.

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

3.2.4.2 Receiving Notification Details

3.2.4.2.1 Receiving RopNotify

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

3.2.5 Timer Events

None.

3.2.6 Other Local Events

None.

4 Protocol Examples

None.

5 Security

5.1 Security Considerations for Implementers

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

6 Appendix A: Office/Exchange Behavior

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

- Office 2003 with Service Pack 3 applied
- Exchange 2003 with Service Pack 2 applied
- Office 2007 with Service Pack 1 applied
- Exchange 2007 with Service Pack 1 applied

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

<1> Section 1.7: Microsoft Office 2007 SP1 and Microsoft Exchange Server 2007 SP1 support Asynchronous RPC Notifications.

Index