[MS-OXCTABL]:
Table Object 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.
04/25/2008	0.2		Revised and updated property names and other technical content.
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		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	3.2.0	Minor	Updated the technical content.

Copyright © 2010 Microsoft Corporation.

Table of Contents

Copyright © 2010 Microsoft Corporation.

Copyright © 2010 Microsoft Corporation.

Release: Friday, February 5, 2010

Copyright © 2010 Microsoft Corporation.

Copyright © 2010 Microsoft Corporation.

1 Introduction

This document specifies the Table Object protocol, which is used by a client to read and navigate through data that is retrieved in tabular format from the server. In addition to retrieving filtered, sorted **rows** of tabular data, the Table Object protocol also allows the client to collapse a grouping of rows and to navigate through the rows.

1.1 Glossary

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

action
attachment
attachments table
binary large object (BLOB)
contents table
flags
folder
Folder object
handle
handle array
header
instance
little-endian
LogonID
message
Message object
permissions
property: (1)
property ID
property set
property tag
property type
restriction
remote operation (ROP)
ROP request buffer
ROP response buffer
rule
rules table
search criteria
single instance
sort order
store
table

The following terms are specific to this document:

bookmark: A data structure that the server uses to point to a position in the table. There are three pre-defined bookmarks (beginning, end, and current). A custom bookmark is a server-specific data structure that can be stored by the client for easily navigating a table.

category: A grouping of rows in a table that all have the same value for a specified property.

column set: A set of properties that are requested by the client for each row of data.

Copyright © 2010 Microsoft Corporation.

Release: Friday, February 5, 2010

cursor: The location of the next row to be read from the table. (The location marked by the cursor is also referred to as the current location in the table.) The cursor may point to any row in the table, or may be located after the last row in the table.

header row: A row at the beginning of a category that does not represent data in the table, but provides information about a grouping.

hierarchy table: A table of folders in a folder.

leaf row: A row that is in a category.

multi-value instance: A row in a table corresponding to a single value in a multi-value property. There will be multiple rows for each Message object in the table, each row corresponding to one value of the multiple value property. Each row will have a single value for the property and the properties for the other columns are repeated.

multi-value property: A property that can have multiple values simultaneously.

permissions table: A table of permissions to a folder.

row: A set of properties associated with one record in the table.

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

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

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

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

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

[MS-OXCPERM] Microsoft Corporation, "Exchange Access and Operation Permissions Protocol Specification", June 2008.

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

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

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

[MS-OXORULE] Microsoft Corporation, "E-Mail Rules Protocol Specification", June 2008.

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

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

1.2.2 Informative References

None.

1.3 Protocol Overview

The Table Object protocol is used to read tabular data from a server. It specifies a set of operations that a client can use to request tabular data from a server based on a **handle** to the **table**. The client can specify the columns, the **restriction**, and the **sort order** for the table, and can request that the rows of the table be categorized according to specific **properties**. The client can then request one or more rows of data. Additionally, the client can find rows, navigate through the rows, and create **bookmarks** for easier navigation. The protocol can provide a way of freeing server **resources** associated with bookmarks.

When the client requests that the rows of the table be categorized, the server will include "**header**" rows in the table which don't have the same properties as normal rows. The client can request that the server hide or show all of the normal rows for which the **header row** represents their **category**. Categories can be nested inside categories. The client can retrieve a **BLOB** that specifies which categories are collapsed and which are expanded in the current table. This can then be given to the table at a future time to restore the collapsed state of the table as well as the **cursor** location.

Multi-value instances can be retrieved from the table when a **multi-value property** is specified in the **column set**. When multi-value instances are requested, for each value in a multi-value property there will be an **instance** of the row that has that single value for the property. All other properties are repeated in each multi-value instance.

Categories that are based on multi-valued properties will display the multi-value instances under each header representing a value that is set on that row. The row that is displayed under a given header row will include the single property value specified by the header row, not all values for the property.

Some tables might not support certain table operations. For example, **rules tables** do not support sorting and return an error if attempted. Tables that do not support asynchronous operations can perform them synchronously or return an error.

1.3.1 Table Notifications

Tables are not static representations of the data. table rows can be modified, moved, created, and deleted while the client is using the table. Table notifications are used to inform the client of all changes made to the table since it was opened.

To properly use the Table Object protocol, both client and server need to implement the Core Notifications protocol, as specified in [MS-OXCNOTIF].

1.4 Relationship to Other Protocols

The Table Object protocol uses the Remote Operations (ROP) List and Encoding Protocol, as specified in [MS-OXCROPS], and the Core Notifications protocol, as specified in [MS-OXCNOTIF].

Copyright © 2010 Microsoft Corporation.

The Message and Attachment object Protocol, the E-mail Rules Protocol, the Exchange Access and Operation Permissions protocol, and the Folder Object Protocol, depend on the Table Object protocol. For information about these protocols, see [MS-OXCMSG], [MS-OXORULE], [MS-OXCPERM], and [MS-OXCFOLD], respectively.

1.5 Prerequisites/Preconditions

The Table Object protocol assumes that the client has acquired a handle to the table object on which it is going to operate. The method by which a handle to a table object is acquired is dependent on the table type. For specifications on how to obtain a handle to a specific table type, see the appropriate document, as stated in the following list:

Table Type	ROP to get a table handle	Specified in
Contents Table	RopGetContentsTable	[MS-OXCFOLD]
Hierarchy Table	RopGetHierarchyTable	[MS-OXCFOLD]
Attachments Table	RopGetAttachmentTable	[MS-OXCMSG]
Permissions Table	RopGetPermissionsTable	[MS-OXCPERM]
Rules Table	RopGetRulesTable	[MS-OXORULE]

1.6 Applicability Statement

The Table Object protocol is used to query tabular data associated with **folders**, **messages**, **attachments**, **permissions**, and **rules** on the server.

1.7 Versioning and Capability Negotiation

The Table Object protocol does not support negotiation of the version to be used. Instead, the client determines the server version to which it has connected. The client's behavior is limited by the capabilities of the server version with which it communicates.

A feature, packed buffers for the RopQueryRows **ROP** (see 3.2.5.5), was added for servers with a major version of eight (8). The client checks the version number that is returned by the server in the results from the **EcDoConnectEx** method, as specified in [MS-OXCRPC] section 3.1.4.11.

1.8 Vendor-Extensible Fields

None.

1.9 Standards Assignments

None.

Copyright © 2010 Microsoft Corporation.

2 Messages

Unless otherwise specified, sizes in this section are expressed in bytes. <1>

2.1 Transport

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

2.2 Message Syntax

2.2.1 Table-Specific Properties

The following properties can be included in the column set of a table for the purpose of collapsing and expanding categories. The following properties are used by the client, but are produced by the server. For more details about these properties, see [MS-OXPROPS].

2.2.1.1 PidTagInstID

This is a **PtypInteger64** property that is an **identifier** for all instances of a row in the table. When a RopGetCollapseState request is sent, the client passes this property value in the **RowId** field to specify a cursor to **store**. When the PidTagInstID property is included in the column set, the server sets the property to the same value for each row that is an instance of the same underlying data.

2.2.1.2 PidTagInstanceNum

This is a **PtypInteger32** property that is an identifier for a **single instance** of a row in the table. When a RopGetCollapseState request is sent, the client passes this property value in the **RowInstanceNumber** field to specify a cursor to store. When this property is included in the column set, the server sets it to a different value for each row that is an instance of the same underlying data.

2.2.1.3 PidTagRowType

This is a **PtypInteger32** property that identifies the type of the row. The possible values are:

Name	Value	Meaning
TBL_LEAF_ROW	0x00000001	The row is a row of data.
TBL_EMPTY_CATEGORY	0x00000002	The row is a header row with no rows inside the category.
TBL_EXPANDED_CATEGORY	0x00000003	The row is a header row that is expanded.
TBL_COLLAPSED_CATEGORY	0x00000004	The row is a header row that is collapsed.

2.2.1.4 PidTagDepth

This is a **PtypInteger32** property that specifies the number of nested categories in which a given row is contained. For example, if a row is contained within two header rows, its depth is 0x0002. When a table contains no categories, all rows will have a depth of 0x0000.

Copyright © 2010 Microsoft Corporation.

2.2.1.5 PidTagContentCount

This is a **PtypInteger32** property that specifies the number of rows under the header row. This property is set whether the header row is collapsed or expanded.

2.2.1.6 PidTagContentUnreadCount

This is a **PtypInteger32** property that specifies the number of rows under the header row that have the PidTagRead **property set** to FALSE. This value is set whether the header row is collapsed or expanded.

2.2.2 Table ROPs

The following sections specify the semantics of ROP fields that are specific to the Table Object protocol. Before sending these requests to the server, the handle to the table object that is used in the **ROP requests** MUST be acquired.

For more details about the syntax for these ROPs, as well as the semantics of common ROP fields, see [MS-OXCROPS].

2.2.2.1 Table ROP Constants

2.2.2.1.1 Pre-Defined Bookmarks

The following values are used in the RopSeekRow request, the RopQueryRows response, and the RopFindRow request.

Name	Value	Meaning
BOOKMARK_BEGINNING	0x00	Points to the beginning position of the table, or the first row.
BOOKMARK_CURRENT	0x01	Points to the current position of the table, or the current row.
BOOKMARK_END	0x02	Points to the ending position of the table, or the location after the last row.

2.2.2.1.2 Custom Bookmarks

The following value is used in the RopFindRow request.

Name	Value	Meaning
BOOKMARK_CUSTOM <2>	0x03	Points to the custom position in the table. Used with the **BookmarkSize** and bookmark fields.

2.2.2.1.3 Table Status

The table status refers to the status of any asynchronous operations being performed on the table. The following values are used in the RopGetStatus, RopAbort, RopSetColumns, RopRestrict, and RopSortTable responses.

Name	Value	Meaning
TBLSTAT_COMPLETE	0x00	No operations are in progress.

Copyright © 2010 Microsoft Corporation.

Release: Friday, February 5, 2010

Name	Value	Meaning
TBLSTAT_SORTING	0x09	A RopSortTable operation is in progress.
TBLSTAT_SORT_ERROR	0x0A	An error occurred during a RopSortTable operation.
TBLSTAT_SETTING_COLS	0x0B	A RopSetColumns operation is in progress.
TBLSTAT_SETCOL_ERROR	0x0D	An error occurred during a RopSetColumns operation.
TBLSTAT_RESTRICTING	0x0E	A RopRestrict operation is in progress.
TBLSTAT_RESTRICT_ERROR	0x0F	An error occurred during a RopRestrict operation.

2.2.2.1.4 Asynchronous Flags

The asynchronous **flags** are flags that specify whether certain ROPs should be performed asynchronously.

Name	Value	Meaning
TBL_ASYNC	0x01	The server SHOULD perform the operation asynchronously. The server can perform the operation synchronously. For details about asynchronous table operations, see sections 3.1.4.1.1 and 3.2.5.1.

2.2.2.2 RopSetColumns Semantics

RopSetColumns is used to set the properties that the client wants to be included in the table. This ROP is only valid on table objects.

2.2.2.2.1 Request Field Overview

2.2.2.2.1.1 SetColumnsFlags

This is a **BYTE** field which contains an OR'ed combination of the asynchronous flags, which are specified in section 2.2.2.1.4. The field MUST NOT have any of the other bits set.

2.2.2.2.1.2 PropertyTagCount

This is a **WORD** field that specifies the number of **property tags** in the **PropertyTags** field. This value MUST be 1 or greater.

2.2.2.2.1.3 PropertyTags

This is an array of PropertyTag structures identifying the set and order of property values to be returned by the server in the ROP response buffer of RopQueryRows, RopFindRow, and RopExpandRow <3>, as specified in sections 2.2.2.5, 2.2.2.14, and 2.2.2.17, respectively. Every table MUST have at least one column. If the **property type** is a multi-valued property, and the client wants multi-value instances based on this property, it MUST also set the **MultivalueInstance** bit (0x2000) of the **PropertyTag** structure. If the property type is not multi-valued, it does not set the **MultivalueInstance** bit of the **PropertyTag** structure set.

For details about the **MultivalueInstance** bit, see [MS-OXCDATA] section 2.13.1.2. For details about the PropertyTag structure, see [MS-OXCDATA] section 2.11.

Copyright © 2010 Microsoft Corporation.

2.2.2.2.2 Response Field Overview

2.2.2.2.2.1 TableStatus

This is a **BYTE** field indicating the status of asynchronous operations being performed on the table. It MUST have one of the table status values that are specified in 2.2.2.1.3.

2.2.2.3 RopSortTable Semantics

RopSortTable orders the rows of a contents table based on sort criteria. This ROP is valid only on table objects.

2.2.2.3.1 Request Field Overview

2.2.2.3.1.1 SortTableFlags

This is a **BYTE** field which contains an OR'ed combination of the asynchronous flags, which are specified in section 2.2.2.1.4. The field MUST NOT have any of the other bits set.

2.2.2.3.1.2 SortOrderCount

This is a **WORD** field that specifies the number of **SortOrder** structures in the **SortOrders** field.

2.2.2.3.1.3 CategoryCount

This is a **WORD** field that specifies the number of **SortOrder** structures in **SortOrders** that are designated as category columns. They occupy the first **CategoryCount** positions in the **SortOrders** array. The value of the **CategoryCount** field MUST be in the range 0 to **SortOrderCount**.

2.2.2.3.1.4 ExpandedCount

This is a **WORD** field that specifies the number of categories that start in the expanded state. This value MUST be in the range 0 to **CategoryCount**. The first **ExpandedCount** categories are initially expanded. If **CategoryCount** is equal to **ExpandedCount**, then all categories are expanded.

2.2.2.3.1.5 SortOrders

This is an array of **SortOrder** structures which defines the sort to be performed. The number of **SortOrder** structures in the array MUST be equal to the value of **SortOrderCount**. For categorized sorting, the **SortOrder** structure specifies the property type and **property ID** that are used as the category. When the value of the **SortOrderCount** field exceeds the value of the **CategoryCount** field, indicating that there are more sort keys than categories, categories are created from the **SortOrder** structures that appear first in the **SortOrders** array. The remaining **SortOrder** structures are used to sort the rows within the categories.

For example, if **SortOrderCount** is set to 0x0003 and **CategoryCount** is set to 0x0002, the columns described by the **PropertyType** and **PropertyId** members of the first two entries in **SortOrders** are used as the category columns. The first entry serves as the top-level category grouping; the second entry as the secondary grouping. All of the rows that match the two category columns are sorted using the sort key defined in the third entry.

If a **SortOrder** structure specifies a multi-value property, it MUST also have the MultivalueInstance bit set, specifying that the sort be performed using the individual values of that property. Sort order on a multi-value property that is not also being used for multi-value instances is undefined. The

Copyright © 2010 Microsoft Corporation.

Release: Friday, February 5, 2010

SortOrders field cannot contain more than one **SortOrder** structure specifying a multi-value property in the first **CategoryCount** positions of the array.

If the **Order** member of a **SortOrder** structure is set to Ascending, the table will be sorted in ascending order by the column specified in the **PropertyType** and **PropertyId** members.

If the **Order** member of a **SortOrder** structure is set to Descending, the table will be sorted in descending order by the column specified in the **PropertyType** and **PropertyId** members.

If the **Order** member of a **SortOrder** structure is set to MaximumCategory, that structure directly follows the first **CategoryCount** structure in the **SortOrders** field and **CategoryCount** MUST be greater than 0x0000 (zero). This **SortOrder** structure, at position **CategoryCount** + 1 in the array, modifies the immediately previous category sort. The categories in the table will be not be sorted by the column specified in the category sort. The categories will be sorted according to each category's maximum value of the specified column. Any **SortOrder** structures after this one will subsort the rows within each category.

For details about the **SortOrder** structure, see [MS-OXCDATA] section 2.14.1.

2.2.2.3.2 Response Field Overview

2.2.2.3.2.1 TableStatus

This is a BYTE field indicating the status of asynchronous operations being performed on the table. It MUST have one of the table status values that are specified in 2.2.2.1.3.

2.2.2.4 RopRestrict Semantics

RopRestrict establishes a restriction on a table. Applying a restriction has no effect on the underlying data of a table; it simply alters the table by limiting the rows that can be retrieved to rows containing data that satisfy the restriction. This ROP is only valid on table objects.

2.2.2.4.1 Request Field Overview

2.2.2.4.1.1 RestrictFlags

This is a **BYTE** field which contains an OR'ed combination of the asynchronous flags, as specified in section 2.2.2.1.4. The field cannot have any of the other bits set.

2.2.2.4.1.2 RestrictionDataSize

This is a **WORD** field that specifies the size, in **BYTES**, of the **RestrictionData** field.

2.2.2.4.1.3 RestrictionData

This is a restriction that is applied to the table. For details about restrictions, see [MS-OXCDATA] section 2.14. It has **RestrictionDataSize BYTES**.

2.2.2.4.2 Response Field Overview

2.2.2.4.2.1 TableStatus

This is a **BYTE** field indicating the status of asynchronous operations being performed on the table. It MUST have one of the table status values that are specified in 2.2.2.1.3.

Copyright © 2010 Microsoft Corporation.

2.2.2.5 RopQueryRows Semantics

RopQueryRows returns zero or more rows from a table, beginning from the current table cursor position. This ROP is only valid on table objects.

2.2.2.5.1 Request Field Overview

2.2.2.5.1.1 QueryRowsFlags

This is a BYTE field which contains either of the following bit flags:

Name	Value	Meaning
Advance	0x00	Advance the table cursor.
NoAdvance	0x01	Do not advance the table cursor.
EnablePackedBuffers	0x02	Enable packed buffers for the response. To allow packed buffers to be used, this flag is used in conjunction with the Chain flag (0x00000004) that is passed in the *pulFlags* parameter of the **EcDoRpcExt2** method. For details about extended-buffer packing, see [MS-OXCRPC]. This flag is only supported against servers with a major version of at least eight (8) as described in section 1.7.

This field cannot contain both flags set simultaneously. The field cannot have any of the other bits set.

2.2.2.5.1.2 ForwardRead

This is a **BYTE** field that specifies the direction in which to retrieve rows. It is set to TRUE (0x01) to read the table forwards. It is set to FALSE (0x00) to read the table backwards. It MUST NOT be set to any other value.

2.2.2.5.1.3 RowCount

This is a **WORD** field that specifies the maximum number of rows to be returned.

2.2.2.5.2 Response Field Overview

2.2.2.5.2.1 Origin

This is a **BYTE** field that identifies the cursor position. It MUST be set to one of the pre-defined bookmark values specified in 2.2.2.1.1.

2.2.2.5.2.2 RowCount

This is a **WORD** field that specifies the number of rows returned. It MUST be less than or equal to the **RowCount** that is specified in the request, and it MUST be greater than or equal to 0x0000. It MUST be equal to the number of **PropertyRow** objects returned in the **RowData** field.

2.2.2.5.2.3 RowData

The array of rows returned. Each row is represented by a PropertyRow object. For details about the format of the PropertyRow object, see [MS-OXCDATA]. Each row MUST have the same columns and

Copyright © 2010 Microsoft Corporation.

ordering of columns as specified in the last RopSetColumns request. The **RowData** field MUST NOT include rows that don't match the criteria specified in the last RopRestrict request. If RopRestrict has not been issued, the **RowData** field MUST include all rows. The rows MUST be sorted and grouped according to the sort order specified in the last RopSortTable request. If RopSortTable has not been sent, the default sort order is undefined. Every property value returned in a row MUST be less than or equal to 510 bytes in size. If a property value is greater than 510 bytes in size, it MUST be truncated to 510 bytes.

2.2.2.6 RopAbort Semantics

RopAbort attempts to stop any asynchronous table operations that are currently in progress. This ROP is only valid on table objects.

2.2.2.6.1 Request Field Overview

There are no ROP-specific request fields.

2.2.2.6.2 Response Field Overview

2.2.2.6.2.1 TableStatus

This is a **BYTE** field indicating the status of asynchronous operations being performed on the table before the abort. It MUST have one of the table status values that are specified in 2.2.2.1.3.

2.2.2.7 RopGetStatus Semantics

The RopGetStatus ROP retrieves information about the current status of asynchronous operations being performed on the table. This ROP is only valid on table objects.

2.2.2.7.1 Request Field Overview

There are no ROP-specific request fields.

2.2.2.7.2 Response Field Overview

2.2.2.7.2.1 TableStatus

This is a **BYTE** field indicating the status of asynchronous operations being performed on the table. It MUST have one of the table status values that are specified in 2.2.2.1.3.

2.2.2.8 RopQueryPosition Semantics

RopQueryPosition returns the location of cursor in the table. Note that the current position and total number of rows could change based on external events before the response to this message is received. This ROP is only valid on table objects.

2.2.2.8.1 Request Field Overview

There are no ROP-specific request fields.

2.2.2.8.2 Response Field Overview

2.2.2.8.2.1 Numerator

This **ULONG** field contains the index (0-based) of the current row. It MUST be greater than or equal to 0x00000000.

2.2.2.8.2.2 Denominator

This **ULONG** field contains the total number of rows in the table. It MUST be greater than or equal to the value of the **Numerator** field.

2.2.2.9 RopSeekRow Semantics

RopSeekRow moves the table cursor to a specific location in the table. The new location is specified by a pre-defined bookmark and the number of rows to move (forward or backwards) from that bookmark. This ROP is only valid on table objects.

2.2.2.9.1 Request Field Overview

2.2.2.9.1.1 Origin

This **BYTE** field contains the bookmark indicating the starting position of the seek operation. This field's value MUST be one of the pre-defined bookmark values that are specified in 2.2.2.1.1.

2.2.2.9.1.2 RowCount

This **LONG** field contains the number of rows to seek, starting from the bookmark. To seek forward from the bookmark, the value MUST be positive; to seek backwards, the value MUST be negative.

2.2.2.9.1.3 WantRowMovedCount

This is a **BYTE** field that specifies whether the actual number of rows moved MUST be returned by the server. This field MUST be set to TRUE (0x01) or FALSE (0x00). If this field is set to TRUE (0x01), the server MUST return the actual number of rows moved.

The actual number of rows moved can differ from the requested number of rows if the beginning or end of the table is encountered prior to moving the requested number of rows.

2.2.2.9.2 Response Field Overview

2.2.2.9.2.1 HasSoughtLess

This is a **BYTE** field that specifies whether the number of rows moved is less than the number of rows requested. This field MUST be set to TRUE (0x01) if the number of rows moved is less than the number of rows requested (**RowCount** field), otherwise it MUST be set to FALSE (0x00).

The **HasSoughtLess** field MUST be present in the response. This field's value MUST be valid if the **WantRowMovedCount** field (in the request) is set to TRUE and its value MUST be ignored if **WantRowMovedCount** is set to FALSE.

Copyright © 2010 Microsoft Corporation.

2.2.2.9.2.2 RowsSought

This **LONG** field specifies the actual number of rows moved. If the value of the **RowCount** field (in the request) is negative, the value of **RowsSought** MUST also be negative or 0x00000000, indicating that the seek was performed backwards.

This field MUST be present in the response. This field's value MUST be valid if the **WantRowMovedCount** field (in the request) is set to TRUE and MUST be ignored if **WantRowMovedCount** is set to FALSE.

2.2.2.10 RopSeekRowBookmark Semantics

RopSeekRowBookmark moves the table cursor to a specific location in the table. The new location is specified by a custom bookmark and the number of rows to move (forward or backwards) from that bookmark.

This ROP is distinguished from RopSeekRow in that the bookmark is not a pre-defined one, but one created by a RopCreateBookmark request. This ROP is only valid on table objects.

2.2.2.10.1 Request Field Overview

2.2.2.10.1.1 BookmarkSize

This is a **WORD** field that specifies the size in bytes of the bookmark field.

2.2.2.10.1.2 Bookmark

A bookmark indicating the starting position of the seek operation. The bookmark MUST be data that was returned by a previous RopCreateBookmark request. The bookmark MUST NOT have been previously freed using RopFreeBookmark<4>.

2.2.2.10.1.3 RowCount

This is a **LONG** field that specifies the number of rows to seek, starting from the bookmark. To seek forward from the bookmark, the value MUST be positive; to seek backward, the value MUST be negative.

2.2.2.10.1.4 WantRowMovedCount

This is a **BYTE** field that specifies whether the actual number of rows moved MUST be returned by the server. This field MUST be set to TRUE (0x01) or FALSE (0x00). If this field is set to TRUE (0x01), the server MUST return the actual number of rows moved.

The actual number of rows moved can differ from the requested number of rows if the beginning or end of the table is encountered prior to moving the requested number of rows.

2.2.2.10.2 Response Field Overview

2.2.2.10.2.1 RowNoLongerVisible

This **BYTE** field indicates whether the row to which the bookmark pointed is no longer visible. This field MUST be set to TRUE (0x01) if the row to which the bookmark pointed has been removed from the table. (For example, the row's properties changed so that they didn't match the restriction, the row was deleted, or the row's header row has been collapsed.) Otherwise, this field MUST be set to FALSE (0x00).

When the row to which the bookmark pointed is no longer visible, the bookmark will **point** to the next row in the table. In this case, the seek will begin from the next row after the bookmark in the table.

2.2.2.10.2.2 HasSoughtLess

This **BYTE** field that specifies whether the number of rows moved is less than the number of rows requested. This field MUST be set to TRUE (0x01) if the number of rows moved is less than the number of rows requested (**RowCount** field), otherwise it MUST be set to FALSE (0x00).

The **HasSoughtLess** field MUST be present in the response. This field's value MUST be valid if the **WantRowMovedCount** field (in the request) is set to TRUE and its value MUST be ignored if **WantRowMovedCount** is set to FALSE.

2.2.2.10.2.3 RowsSought

This **LONG** field specifies the actual number of rows moved. If the value of the **RowCount** field (in the request) is negative, the value of **RowsSought** MUST also be negative or 0x00000000, indicating that the seek was performed backwards.

This field MUST be present in the response. This field's value MUST be valid if the **WantRowMovedCount** field (in the request) is set to TRUE and MUST be ignored if **WantRowMovedCount** is set to FALSE.

2.2.2.11 RopSeekRowFractional Semantics

RopSeekRowFractional moves the table cursor to an approximate position in the table. The new location is specified as a faction of the table size. This ROP is only valid on table objects.

2.2.2.11.1 Request Field Overview

2.2.2.11.1.1 Numerator

This **ULONG** field is the numerator of the fractional position. The value MUST be less than or equal to the value of the **Denominator** field. If the value of **Numerator** is 0x00000000, the cursor MUST be set to the first row in the table. If the value is equal to or greater than the value of **Denominator**, the cursor MUST be set past the last row in the table.

2.2.2.11.1.2 Denominator

This **ULONG** field is the denominator of the fractional position. The value MUST NOT be set to 0x00000000.

2.2.2.11.2 Response Field Overview

There are no ROP-specific response fields.

2.2.2.12 RopCreateBookmark Semantics

RopCreateBookmark creates a new bookmark at the current cursor position in the table. This ROP is only valid on table objects.

2.2.2.12.1 Request Field Overview

There are no ROP-specific request fields.

2.2.2.12.2 Response Field Overview

2.2.2.12.2.1 BookmarkSize

This **WORD** field specifies the size in bytes of the bookmark field. This field MUST be present.

2.2.2.12.2.2 Bookmark

This field contains the bookmark data. This data is specific to the server. The client MUST NOT assume that this data has a specific format.

2.2.2.13 RopQueryColumnsAll Semantics

RopQueryColumnsAll returns a complete list of all columns for the table. The list includes all columns that the server has for the table, not necessarily only those requested by RopSetColumns. This ROP is only valid on table objects.

2.2.2.13.1 Request Field Overview

There are no ROP-specific request fields.

2.2.2.13.2 Response Field Overview

2.2.2.13.2.1 PropertyTagCount

This **WORD** field specifies the number of property tags in the **PropertyTags** field.

2.2.2.13.2.2 PropertyTags

An array of property tags, each of which corresponds to an available column in the table. Each property tag is represented by a **PropertyTag** structure. For details about the **PropertyTag** structure, see [MS-OXCDATA] section 2.11.

2.2.2.14 RopFindRow Semantics

RopFindRow returns the next row in a table that matches the **search criteria** and moves the cursor to that row. The initial location for the search is specified by a bookmark. This ROP is only valid on table objects.

2.2.2.14.1 Request Field Overview

2.2.2.14.1.1 FindRowFlags

This is a **BYTE** field which contains an OR'ed combination of any of the following bit flags.

Value	Meaning
0x00	Perform the find forwards.
0x01	Perform the find backwards.

Copyright © 2010 Microsoft Corporation.

The field MUST NOT have any of the other bits set.

2.2.2.14.1.2 RestrictionDataSize

This **WORD** field specifies the size, in bytes, of the **RestrictionData** field.

2.2.2.14.1.3 RestrictionData

A restriction which specifies the criteria to be used for the search. For details about restrictions, see [MS-OXCDATA] section 2.14.

2.2.2.14.1.4 Origin

This **BYTE** field specifies the bookmark and MUST be set to either one of the pre-defined bookmark values or BOOKMARK_CUSTOM. For details about bookmarks, see sections 2.2.2.1.1 and 2.2.2.1.2.

2.2.2.14.1.5 BookmarkSize

This **WORD** field specifies the size, in **BYTES**, of the bookmark field. If the **Origin** field specifies a pre-defined bookmark, this field MUST be set to 0x0000.

2.2.2.14.1.6 Bookmark

This field specifies the bookmark from which to begin the search. The bookmark MUST be one that was returned by a previous RopCreateBookmark request. The bookmark MUST NOT have been previously freed using RopFreeBookmark. If the **Origin** field specifies a pre-defined bookmark, this field MUST NOT be present in the request.

2.2.2.14.2 Response Field Overview

2.2.2.14.2.1 RowNoLongerVisible

This **BYTE** field indicates whether the row to which the bookmark pointed, is no longer visible. This field MUST be set to TRUE (0x01) if the row to which the bookmark pointed, has been removed from the table. (For example, the row's properties changed so that it didn't match the restriction, the row was deleted, or the row's header row has been collapsed.) Otherwise, this field MUST be set to FALSE (0x00).

When the row to which the bookmark pointed is no longer visible, the search will begin from the next row after the bookmark in the table. When searching backward, the search will not consider the row to which the bookmark is currently pointing, but will begin at the previous row.

2.2.2.14.2.2 HasRowData

This is a **BYTE** field that specifies whether a row meeting the specified criteria was found. If a row that meets the specified criteria was found, this field MUST be set to TRUE (0x01).

2.2.2.14.2.3 RowData

This field contains a **PropertyRow** structure that specifies the row. If the value of the **HasRowData** field is TRUE, the **RowData** field MUST be present. If the value of **HasRowData** is FALSE, the **RowData** field MUST NOT be present.

For details about the **PropertyRow** structure, see [MS-OXCDATA] section 2.9.1.

Copyright © 2010 Microsoft Corporation.

2.2.2.15 RopFreeBookmark Semantics

RopFreeBookmark frees the memory associated with a bookmark that was returned by a previous RopCreateBookmark request. After the bookmark has been released, attempts to use the bookmark will fail with ecInvalidBookmark. This ROP is only valid on table objects.

2.2.2.15.1 Request Field Overview

2.2.2.15.1.1 BookmarkSize

This **WORD** field specifies the size, in **BYTES**, of the bookmark field.

2.2.2.15.1.2 Bookmark

This field specifies the bookmark to be freed. The bookmark MUST be one that was returned by a previous RopCreateBookmark request. The bookmark MUST NOT have been previously freed using RopFreeBookmark.

2.2.2.15.2 Response Field Overview

There are no ROP-specific response Fields.

2.2.2.16 RopResetTable Semantics

RopResetTable performs the following **actions**:

- Removes the existing column set, restriction, and sort order from the table.
- Invalidates bookmarks.
- Resets the cursor to the beginning of the table.

This ROP is only valid on table objects.

After sending this ROP, a RopSetColumns request MUST be sent prior to sending a RopFindRow, RopQueryRows, or RopExpandColumn request. Existing bookmarks SHOULD be freed using RopFreeBookmark. The client can choose not to send RopFreeBookmark, however, this can degrade server performance until the table is released by using RopRelease. For details about RopRelease, see [MS-OXCROPS] section 2.2.14.3.

2.2.2.16.1 Request Field Overview

There are no ROP-specific request fields.

2.2.2.16.2 Response Field Overview

There are no ROP-specific response fields.

2.2.2.17 RopExpandRow Semantics

RopExpandRow expands a collapsed category of a table and returns the rows that belong in the newly expanded category. The maximum number of **leaf rows** to be returned can be specified. This ROP is only valid on table objects.

2.2.2.17.1 Request Field Overview

2.2.2.17.1.1 MaxRowCount

This **WORD** field specifies the maximum number of leaf rows to be returned in the response.

2.2.2.17.1.2 CategoryId

This 8-**BYTE** field specifies the category to be expanded. This field is set to the value of the PidTagInstID property of the category's header row.

2.2.2.17.2 Response Field Overview

2.2.2.17.2.1 ExpandedRowCount

This **ULONG** field specifies the total number of rows that are in the expanded category.

2.2.2.17.2.2 RowCount

This **WORD** field specifies the number of **PropertyRow** structures that are contained in the **RowData** field. The value of this field MUST be less than or equal to both of the following:

- The value of the **MaxRowCount** field in the request buffer.
- The value of the **ExpandedRowCount** field in the response buffer.

2.2.2.17.2.3 RowData

This field contains an array of **PropertyRow** structures, each of which specifies a row in the expanded category. For details about the **PropertyRow** structure, see [MS-OXCDATA] section 2.9.1.

2.2.2.18 RopCollapseRow Semantics

RopCollapseRow <5> collapses an expanded category. This ROP is only valid on table objects.

2.2.2.18.1 Request Field Overview

2.2.2.18.1.1 CategoryId

This 8-**BYTE** field specifies the category to be collapsed. This field is set to the value of the PidTagInstID property of the category's header row.

2.2.2.18.2 Response Field Overview

2.2.2.18.2.1 CollapsedRowCount

This **ULONG** field specifies the number of rows that have been collapsed.

2.2.2.19 RopGetCollapseState Semantics

RopGetCollapseState <6>returns the data necessary to rebuild the current expanded/collapsed state of the table. The data returned is in the form of an opaque BLOB that can be passed to a RopSetCollapseState request. This ROP is only valid on table objects.

Copyright © 2010 Microsoft Corporation.

Release: Friday, February 5, 2010

2.2.2.19.1 Request Field Overview

2.2.2.19.1.1 RowId

This 8-**BYTE** field specifies the row to be preserved as the current cursor, which is returned in the **CollapseState** field of the RopGetCollapseState response. This field is set to the value of the PidTagInstID property of the row to be preserved as the current cursor.

2.2.2.19.1.2 RowInstanceNumber

This **ULONG** field is set to the value of the PidTagInstanceNum property of the row to be preserved as the current cursor, which is returned in the **CollapseState** field of the RopGetCollapseState response.

2.2.2.19.2 Response Field Overview

2.2.2.19.2.1 CollapseStateSize

This **WORD** field specifies the size, in bytes, of the **CollapseState** field.

2.2.2.19.2.2 CollapseState

This field contains the data necessary for RopSetCollapseState <7> to rebuild the table's collapsed state, including the current cursor.

2.2.2.20 RopSetCollapseState Semantics

RopSetCollapseState rebuilds a table's collapsed state, which is specified by the data returned from RopGetCollapseState.The RopSetCollapseState response contains a bookmark referencing the row that was identified by the **RowId** and **RowInstanceNumber** fields in the RopGetCollapseState request. This ROP is only valid on table Objects.

The collapsed state sent to the server need not have been retrieved from the same table to which it is being applied. The table MUST have the same sort and restriction for the ROP to succeed. If the table is not the same table from which the collapse state was retrieved, the bookmark specified in the RopSetCollapseState response will be invalid. The bookmark returned MUST be freed by using RopFreeBookmark.

2.2.2.20.1 Request Field Overview

2.2.2.20.1.1 CollapseStateSize

This **WORD** field specifies the size, in bytes, of the **CollapseState** field.

2.2.2.20.1.2 CollapseState

This field contains the data that is necessary to rebuild the table's collapsed state. This data is obtained by sending a RopGetCollapseState request.

2.2.2.20.2 Response Field Overview

2.2.2.20.2.1 BookmarkSize

This **WORD** field specifies the size, in **BYTES**, of the bookmark field.

2.2.2.20.2.2 Bookmark

This field contains the bookmark data for the row stored as the current cursor, which was passed in the **CollapseState** field of the RopSetCollapseState request.

Copyright © 2010 Microsoft Corporation.

3 Protocol Details

3.1 Client Details

A table is not a static representation of data. Table rows can be modified, moved, created, and deleted while the table object is in use. Table notifications are used to inform the client of all changes made to the table since it was opened.

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 document does not mandate that implementations adhere to this model as long as their external behavior is consistent with that described in this document.

Row: A set of properties associated with one record in the table.

Cursor: Represents the current location in the table. A table cursor refers to a data row, not an ordinal row within the table, so if the client were to read 50 rows from the table and then a row was inserted above the rows that were read, the next query will not read the same rows again. The server's table cursor keeps its position associated with the data row that was directly below the last row returned. If the current row is deleted, the server will reposition the cursor to the next row at the time of deletion. The cursor is reset to the beginning of the table when the sort order or restriction on the table changes.

Bookmark: Points to a row in the table. The client can request that the table create a bookmark at the current location. This allows the client to save the cursor location in a table and restore it when needed. One common usage is to store the cursor location while getting data from elsewhere in the table. There are three pre-defined bookmarks in every table. These are the beginning of the table, the end of the table, and the current location. Bookmarks refer to data rows, just as the cursor does, and not ordinal rows. If the row pointed to by a bookmark is deleted, the server will reposition the bookmark to the next row at the time of deletion. Bookmarks are invalidated when the sort order or restriction on the table changes. When invalid bookmarks are used in the protocol, the request will fail. Invalid bookmarks still need to be freed.

Sort Order: Determines the order in which rows are returned. Categories can be sorted using an aggregation property. For example, messages can be categorized by **conversation**, and then categories are sorted based on received time (for example, the conversation with the most recent message will appear first in the table).

Restriction: Filters out rows which do not meet a given criteria.

Column Set: Determines which property values are returned for each row.

Category: Tables can categorize rows by column values. A category is a grouping of rows in a table which all have the same value for a specific column. Each category is preceded by a header row that identifies the common property value. Categories can be nested such that rows within a category are then grouped by another property (for received date and then sender). Each category in this sub-category is also preceded by a header row. The client can request the rows of a category be collapsed. When a category is collapsed, all the rows, including sub-category header rows are removed from the table, leaving only the category's header row. A collapsed category can then be expanded to reveal the rows within the category. Categories are only supported on contents tables.

Multi-Value Instance: When a table includes multi-value properties, it can have multi-value instances, or multiple rows for each underlying row of data, each row corresponding to a single value in the multi-value property. If a table includes multi-value instances, it can also be categorized based on the multi-value property. A given message can be placed in multiple categories, based on the individual values of the multi-valued property.

A client can have more than one table opened on each data source. Each table is independent and has its own cursor, bookmarks, sort order, restriction, column set, and notification requests.

3.1.2 Timers

There are no timers specific to this protocol.

3.1.3 Initialization

A client can begin using this protocol with a valid table handle. The method to open the table and acquire a handle is dependent on the table's type. The types of tables are specified in section 1.5.

3.1.4 Higher-Layer Triggered Events

3.1.4.1 Preparing the Table

When a higher layer, or the user, needs to modify the column set, the client MUST send RopSetColumns to change the column set.

When a higher layer, or the user, needs to modify the sort order, the client MUST send RopSortTable to change the sort order. RopSortTable is only supported on contents tables.

In order to categorize a table, the category properties MUST be the first properties in the **SortOrders** array passed to RopSortTable. The number of category properties MUST be set in the **CategoryCount** field passed to RopSortTable.

When a higher layer, or the user, needs to filter the rows returned in RopQueryRows, RopFindRow, or RopExpandRow, the client MUST send RopRestrict to change the filter applied to the table.

When a higher layer, or the user, needs to clean up the table by removing old sorts, column sets or restrictions, the client MUST send RopResetTable. After doing this, the client MUST send RopSetColumns before sending RopQueryRows, RopFindRow, or RopExpandRow.

If a request to RopSetColumns, RopSortTable, or RopRestrict fails, the client can send RopResetTable before retrying the failed message.

If RopSetColumns fails, the client MUST consider the table as invalid and MUST NOT send any other ROPs on it until a successful RopSetColumns or RopResetTable is made.

If RopSortTable fails, the client MUST consider the table as invalid and MUST NOT send any ROPs on it until a successful RopSortTable or RopResetTable is made.

If RopRestrict fails, the client MUST consider the table as invalid and MUST NOT send any ROPs on it until a successful RopRestrict or RopResetTable is made.

3.1.4.1.1 Asynchronous Table Preparation

The client MAY request that the server perform RopSetColumns, RopSortTable, or RopRestrict asynchronously <8>.In this case, the client MUST NOT request additional asynchronous work to be done until pending asynchronous work is complete or cancelled using RopAbort <9>. If the client

Copyright © 2010 Microsoft Corporation.

requests additional synchronous work while an existing asynchronous request is pending, the server will respond in one of two ways:

The server returns ecBusy and does not perform the requested action.

The server waits until the first asynchronous action is complete, then completes the synchronous action and sends the **ROP response** at that time.

When a higher layer or the user needs to know the status of pending asynchronous requests, the client MUST get the status using RopGetStatus <10>. When a higher layer, or the user, needs to abort a pending asynchronous request (to set columns, sort the table, or restrict), the client MUST send a RopAbort request. After a successful RopAbort, the client MUST assume the table is in an undefined state and use RopResetTable before using the table again.

If an asynchronous request to RopSetColumns, RopSortTable, or RopRestrict fails with the error ecNotSupported, the client can reattempt the request synchronously.

If a RopSetColumns request fails, the client MUST assume that the table has an invalid column set and MUST perform a successful RopSetColumns before proceeding.

If a RopSortTable request fails, the client MUST assume that the table has an invalid sort and MUST perform a successful RopSortTable before proceeding.

If a RopRestrict request fails, the client MUST assume that the table has an invalid restriction and MUST perform a successful RopRestrict before proceeding.

3.1.4.2 Querying the Table

When a higher layer or the user requests tabular data from a table, the client MUST retrieve that information using RopQueryRows or RopFindRow.

The client can get the whole table by sending RopQueryRows repeatedly with the *Advance* option until RopQueryRows returns zero rows (indicating the end of the table has been reached).

When a higher layer or the user needs to know the list of available columns for the table, the client MUST get the column list by sending RopQueryColumnsAll <11>.

When a higher layer or the user needs to collapse rows that are grouped into a category into one header row, the client MUST send RopCollapseRow.

When a higher layer or the user needs to expand rows that are grouped into a collapsed header row, the client MUST send RopExpandRow. The client can retrieve some or all of the rows expanded.

If the client is going to expand and collapse categories, it MUST include the PidTagInstID property in the PropertyTags field of the RopSetColumns request, and use the value of that property in RopExpandRow, RopCollapseRow, and RopGetCollapseState requests.

3.1.4.3 Advancing the Table

When querying the table, the client can advance the table by setting the **QueryRowsFlags** field to 0x00 in the RopQueryRows request. Additionally, when higher layers need to move the current cursor in the table, the client MUST use RopFindRow, RopSeekRow, RopSeekRowBookmark <12>, or RopSeekRowFractional <13> to advance to the correct row. RopFindRow can be used to both advance the table and query for the row found at the same time.

The client MUST NOT expect RopSeekRowFractional to place the cursor in any exact position. It is always an approximation.

Copyright © 2010 Microsoft Corporation.

Release: Friday, February 5, 2010

When a higher layer or the user needs to determine the current location in a table, the client MUST send the RopQueryPosition request.

When a higher layer or the user needs to save the current location in the table for future use, the client MUST send RopCreateBookmark <14>, and cache the bookmark data.

When a higher layer, or the user, no longer needs a bookmark that was created using RopCreateBookmark, the client SHOULD send RopFreeBookmark with the bookmark data. The client can choose not to send RopFreeBookmark; however, this can degrade server performance until the table is released via RopRelease.

When a higher layer or the user needs to move the current table location to a previously created bookmark, the client MUST send RopSeekRowBookmark.

3.1.4.4 Getting Table State

When a higher layer or the user needs to preserve the expanded/collapsed state of the categories, it MUST send RopGetCollapseState and store the BLOB sent in the response for future use.

When a higher layer or the user needs to reset the expanded/collapsed state of the categories to a previously cached state, it MUST send RopSetCollapseState with the BLOB returned from RopGetCollapseState. The bookmark sent in the response to RopSetCollapseState SHOULD be freed using RopFreeBookmark. The client can choose not to send RopFreeBookmark, however, this can degrade server performance until the table is released via RopRelease.

3.1.4.5 Registering For Notifications

The protocol assumes that tables are dynamic. This means that the state on the server can significantly change while waiting for responses. If the server supports notifications on the table, the client SHOULD register for notifications and respond appropriately in order to have an accurate understanding of server state. See [MS-OXCNOTIF] for details about notifications.

3.1.5 Message Processing Events and Sequencing Rules

After opening the table or sending a RopResetTable request, the client MUST send a RopSetColumns request before querying the table for data.

The client SHOULD send a RopSortTable request before querying a contents table for data. If the client does not send a RopSortTable, it MUST consider the sort order of the table as undefined.

The client can send RopRestrict before querying the table for data.

When the **TableStatus** field of the RopSortTable response has a value that is equal to TBLSTAT_SORT_ERROR, then RopSortTable failed, and the client MUST consider the table invalid until it receives a successful RopSortTable response. When the **TableStatus** field of the RopSetColumns response has a value that is equal to TBLSTAT_SETCOL_ERROR, then RopSetColumns failed, and the client MUST consider the table invalid until it receives a successful RopSetColumns response. When the **TableStatus** field of the RopRestrict response has a value that is equal to TBLSTAT_RESTRICT_ERROR, then RopRestrict failed, and the client MUST consider the table invalid until it receives a successful RopRestrict response.

The client SHOULD NOT send RopAbort unless the last **TableStatus** returned in a ROP response indicated that the server is executing an asynchronous task. If the server has no asynchronous work executing when RopAbort is requested, it will return ecUnableToAbort in the **ReturnValue** field of the RopAbort response.

3.1.5.1 Processing Notifications

The protocol assumes that tables are dynamic. This means that the state on the server can significantly change while waiting for responses. If the server supports notifications on the table, the client SHOULD register for notifications and respond appropriately in order to have an accurate understanding of server state. See [MS-OXCNOTIF] for details about notifications.

3.1.6 Timer Events

There are no timers specific to this protocol.

3.1.7 Other Local Events

There are no other local events specific to this protocol.

3.2 Server 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 document does not mandate that implementations adhere to this model as long as their external behavior is consistent with that described in this document.

The abstract data model for the server is the same as the abstract data model for the client, as defined in section 3.1.1.

3.2.2 Timers

There are no timers specific to this protocol.

3.2.3 Initialization

There is no initialization specific to this protocol.

3.2.4 Higher-Layer Triggered Events

The server MUST send notifications to all clients that have requested them based on the appropriate triggers at higher layers. See [MS-OXCNOTIF] for details on notifications.

3.2.5 Message Processing Events and Sequencing Rules

3.2.5.1 Processing Asynchronous Requests

If the client requests that the server perform RopSetColumns, RopSortTable, or RopRestrict asynchronously, the server can perform the operation synchronously and return TBLSTAT_COMPLETE in the **TableStatus** field of the response buffer. However, the server SHOULD return TBLSTAT_SORTING, TBLSTAT_SETTING_COLS, or TBLSTAT_RESTRICTING (depending on the ROP performed) in the **TableStatus** field of the response buffer, and do the work asynchronously. The server MUST return the same *TableStatus* value in the RopGetStatus response, unless the work has been completed or RopAbort has been sent.

Copyright © 2010 Microsoft Corporation.

Release: Friday, February 5, 2010

If there is an error setting the columns, sorting the table, or restricting, the server MUST send the **TableError** notification. The next response to RopGetStatus MUST set the *TableStatus* field to TBLSTAT_SETCOL_ERROR, TBLSTAT_SORT_ERROR, or TBLSTAT_RESTRICT_ERROR, depending on the ROP performed. When the asynchronous work is complete, the server MUST send the **TableSortDone**, **TableRestrictionChanged**, or **TableColumnsChanged** notifications, depending on the ROP performed. For details about these notifications, see [MS-OXCNOTIF] section 2.2.1.1.1.

If the client requests additional asynchronous work while the server is still performing asynchronous work, the server MUST set the value of the *ReturnValue* field in the response buffer to ecBusy. If the client requests additional synchronous work while the server is still performing asynchronous work, the server can:

Set the value of the *ReturnValue* field in the response buffer to ecBusy and not perform the requested action.

Wait until the first asynchronous action is complete, then complete the synchronous action and send the ROP response at that time.

3.2.5.2 Processing RopSetColumns

The server MUST remember the requested columns, and apply them to the table when executing other ROPs that act on that table. The columns that are set by RopSetColumns MUST be the ones sent in the responses to subsequent RopQueryRows, RopFindRow, or RopExpandRow executed on that table.

If either RopQueryRows or RopFindRow is sent prior to a successful RopSetColumns, then the server MUST fail the ROP with ecNullObject.

If RopSetColumns fails, the server SHOULD invalidate the table column set until a successful RopSetColumns is made. The server can restore the previous column set.

If the TBL_ASYNC bit of the **SetColumnsFlags** field is set, the server can execute the ROP as a table-asynchronous ROP. See section 3.2.5.1.

RopSetColumns MUST be supported for all types of tables.

If a column has the **MultivalueInstance** bit set in a **PropertyTag** structure, the server MUST expand the rows that have multiple values for the property into multi-value instances in subsequent RopQueryRows, RopFindRow, or RopExpandRow that are executed on the table.

The following specific error codes apply to this ROP. For more details about ROP errors returned, see Error Codes in [MS-OXCDATA] section 2.4.

Name	Meaning
ecInvalidParam	A property tag in the column array is of type PT_UNSPECIFIED, PT_ERROR, or an invalid type. These property types are specified in [MS-OXCDATA] section 2.12.1.
ecNotSupported	If the object on which this ROP was sent is not of type table.

3.2.5.3 Processing RopSortTable

The server MUST apply the sort order to the table, and subsequent requests sent that operate on the table MUST consider the new sort order.

Copyright © 2010 Microsoft Corporation.

Release: Friday, February 5, 2010

If a sort order was already specified, the new sort order sent in with the ROP MUST completely replace the old sort order.

When this ROP is sent, the server MUST invalidate all current bookmarks of the table and MUST move the cursor position to the beginning of the table.

If RopSortTable is not sent (a sort order is not specified) then the table MUST be considered as having the default sort order. Default sort order is undefined.

If RopSortTable fails, the server SHOULD invalidate the table sort order until a successful RopSortTable is made. The server can restore the previous sort order.

If the TBL_ASYNC bit of the **SortTableFlags** field is set, the server can execute the ROP as a table-asynchronous ROP. See section 3.2.5.3.

RopSortTable MUST be supported for contents tables.

If a multi-value property has the *MultivalueInstance* bit set in the **SortOrder** structure, the server MUST sort the rows that have multiple values for the property according to the single values used in the multi-value instances in subsequent RopQueryRows, RopFindRow, or RopExpandRow that are executed on the table.

The following specific error codes apply to this ROP. For more details about ROP errors returned, see Error Codes in [MS-OXCDATA] section 2.4.

Name	Meaning
ecNotSupported	If the object on which this ROP was sent is not a contents table.

3.2.5.4 Processing RopRestrict

The server MUST apply the restriction to the table, and subsequent requests that operate on the table MUST consider the new restriction.

If a restriction is applied to a table, the table MUST appear as if it only contains the rows that match the restriction.

When this ROP is sent, the server MUST invalidate all current bookmarks of the table and MUST move the cursor position to the beginning of the table.

If RopRestrict is not sent (a restriction is not specified) then the table MUST be considered as not having any restriction.

If RopRestrict fails, the server SHOULD invalidate the table restriction until a successful RopRestrict is made. The server can restore the previous restriction.

If the TBL_ASYNC bit of the **RestrictFlags** field is set, the server can execute the ROP as a table-asynchronous ROP. See section 3.2.5.3.

RopRestrict MUST be supported for contents tables, hierarchy tables, and rules tables.

The following specific error codes apply to this ROP. For more details about ROP errors returned, see Error Codes in [MS-OXCDATA] section 2.4.

Name	Meaning
ecNotSupported	If the object on which this ROP was sent is not a contents or hierarchy table.

3.2.5.5 Processing RopQueryRows

When RopQueryRows is sent the server MUST send the rows from the table starting at the current cursor position.

The number of rows sent in the response MUST be less than or equal to the number of rows specified on the **RowCount** field. The number of rows sent in the response MUST be as many rows as can fit in the ROP response buffer. Whole rows MUST always be sent (partial rows MUST NOT be sent). If there are rows to send in the database, then at least one row MUST be returned, or the ROP MUST fail with ecBufferTooSmall ([MS-OXCDATA] section 2.4).. This ROP MUST only send zero rows when there are no more rows in the table.

If the **ForwardRead** field is TRUE, RopQueryRows MUST return the rows beginning at the origin, reading forward. If it is FALSE the server MUST return the rows starting at **RowCount** rows before the current cursor position, such that the **RowCount** rows before the cursor position are returned.

If the NoAdvance flag (0x01) is set in the **QueryRowsFlags** field, the server MUST NOT change the position of the cursor.

RopSetColumns MUST be sent on the table prior to sending RopQueryRows. The columns sent in the response for each row MUST be the ones specified on RopSetColumns.

The server MUST complete all asynchronous table ROPs prior to executing this ROP or fail the ROP with ecBusy. See section 3.2.5.1.

RopQueryRows MUST be supported for all types of tables.

The following specific error codes apply to this ROP. For more details about ROP errors returned, see Error Codes in [MS-OXCDATA] section 2.4.

Name	Meaning
ecNullObject	RopSetColumns has not been sent on this table.
ecNotSupported	If the object on which this ROP was sent is not of type table.

3.2.5.6 Processing RopAbort

RopAbort MUST abort the current asynchronous table ROP that is executing on the table, or send an error if there is nothing to abort or if it fails to abort.

If the server receives a RopAbort while asynchronous work is being done, it MUST abort that work. The table state after a RopAbort request is received is undefined until the server receives a RopResetTable request. This is true whether or not the RopAbort succeeds. See section 3.2.5.1.

RopAbort MUST be supported for contents tables and hierarchy tables.

The following specific error codes apply to this ROP. For more details about ROP errors returned, see Error Codes in [MS-OXCDATA] section 2.4.

Name	Meaning
ecUnableToAbort	There were no asynchronous operations to abort, or the server was unable to abort the operations.
ecNotSupported	The object on which this ROP was sent is not a contents table or a hierarchy table.

3.2.5.7 Processing RopGetStatus

RopGetStatus MUST send the status of the current asynchronous execution being performed on the table in the response. See section 3.2.5.1.

RopGetStatus MUST be supported for all types of tables.

The following specific error codes apply to this ROP. For more details about ROP errors returned, see Error Codes in [MS-OXCDATA] section 2.4.

Name	Meaning
ecNotSupported	If the object on which this ROP was sent is not of type table.

3.2.5.8 Processing RopQueryPosition

RopQueryPosition MUST send the current position of the cursor, and the total number of rows in the table in the response.

The server MUST complete all asynchronous table ROPs prior to executing this ROP or fail the ROP with ecBusy. For details, see section 3.2.5.1.

RopQueryPosition MUST be supported for all types of tables.

The following specific error codes apply to this ROP. For more details about ROP errors returned, see Error Codes in [MS-OXCDATA] section 2.4.

Name	Meaning
ecNotSupported	If the object on which this ROP was sent is not of type table.

3.2.5.9 Processing RopSeekRow

RopSeekRow MUST move the cursor position according to its request fields. If moving the cursor **RowCount** rows would put it past the end (or beginning, if seeking backwards) of the table, and the **WantRowMovedCount** field was TRUE in the request, the server MUST set the **HasSoughtLess** field to TRUE and set **RowsSought** to the actual number of rows moved to reach the end of the table.

The server MUST complete all asynchronous table ROPs prior to executing this ROP or fail the ROP with ecBusy. See section 3.2.5.1.

RopSeekRow MUST be supported for all types of tables.

The following specific error codes apply to this ROP. For more details about ROP errors returned, see Error Codes in [MS-OXCDATA] section 2.4.

Name	Meaning
ecNotSupported	If the object on which this ROP was sent is not of type table.

3.2.5.10 Processing RopSeekRowBookmark

RopSeekRowBookmark MUST move the cursor position according to its request fields. It should act in the same way as RopSeekRow, except that it moves the cursor using a custom-defined bookmark as a reference.

If the bookmark has become invalid because of a RopSortTable, RopRestrict, or RopResetTable, the server MUST set the **ReturnValue** field to ecNotFound. If the bookmark points to a row that is no longer visible (for example, it has been deleted, or its properties have changed so that it no longer matches the restriction, or its header row has been collapsed) the server MUST set **RowNoLongerVisible** to TRUE, and move the cursor to the next row in the table.

The server MUST complete all asynchronous table ROPs prior to executing this ROP or fail the ROP with ecBusy. See section 3.2.5.1.

RopSeekRowBookmark MUST be supported for contents tables and hierarchy tables.

The following specific error codes apply to this ROP. For more details about ROP errors returned, see Error Codes in [MS-OXCDATA] section 2.4.

Name	Meaning
ecNotFound	If the bookmark sent in the request is no longer valid.
ecNotSupported	If the object on which this ROP was sent is not a contents or hierarchy table.

3.2.5.11 Processing RopSeekRowFractional

RopSeekRowFractional MUST move the cursor position in the same way as RopSeekRow does, except that the desired position is indicated as a fraction of the total table size.

If the numerator is 0, the cursor MUST move to the beginning of the table.

If the numerator is greater than or equal to the denominator, the cursor MUST move to the end of the table.

The cursor MUST be moved to the place in the table that would be closest to the fraction provided. The exact location is dependent on the implementation of the server.

The server MUST complete all asynchronous table ROPs prior to executing this ROP or fail the ROP with ecBusy. See section 3.2.5.1.

RopSeekRowFractional MUST be supported for all types of tables.

The following specific error codes apply to this ROP. For more details about ROP errors returned, see Error Codes in [MS-OXCDATA] section 2.4.

Name	Meaning
ecNotSupported	If the object on which this ROP was sent is not of type table.

Copyright © 2010 Microsoft Corporation.

Release: Friday, February 5, 2010

3.2.5.12 Processing RopCreateBookmark

The protocol server MUST create a custom bookmark that uniquely identifies a row in the table and can be subsequently used in RopSeekRowBookmark.

The server can allocate resources on the server to keep track of the bookmark created by RopCreateBookmark. If the client does not send RopFreeBookmark, the server MUST release all bookmarks related to a table when that table is released (as a result of sending ROPRelease on the table) or when RopResetTable, RopSortTable, or RopRestrict is sent.

The server MUST complete all asynchronous table ROPs prior to executing this ROP or fail the ROP with ecBusy. See section 3.2.5.1.

RopCreateBookmark MUST be supported for contents tables and hierarchy tables.

The following specific error codes apply to this ROP. For more details about ROP errors returned, see Error Codes in [MS-OXCDATA] section 2.4.

Name	Meaning
ecNotSupported	If the object on which this ROP was sent is not a contents or hierarchy table.

3.2.5.13 Processing RopQueryColumnsAll

RopQueryColumnsAll MUST send all properties that can be queried for in the table in the response.

RopQueryColumnsAll MUST be supported for all types of tables.

The following specific error codes apply to this ROP. For more details about ROP errors returned, see Error Codes in [MS-OXCDATA] section 2.4.

Name	Meaning
ecNotSupported	If the object on which this ROP was sent is not of type table.

3.2.5.14 Processing RopFindRow

RopFindRow MUST move the cursor position to the first row that matches the criteria specified in the ROP (starting the search on the current cursor position) and it MUST send that row. The rows that do not match the criteria MUST continue to be in the table, but the cursor MUST be moved to the first row that matches the criteria. If from the current position, there are no rows that match the criteria, then RopFindRow MUST NOT move. In this case, the cursor position MUST remain as it was prior to the call to RopFindRow.

If the client requested that the find be performed from a custom bookmark, but the bookmark has become invalid because of a RopSortTable, RopRestrict, or RopResetTable, then the server MUST set the **ReturnValue** field to ecNotFound. If the bookmark points to a row that is no longer visible (for example, the row has been deleted, or its properties have changed so that it no longer matches the restriction, or its header row has been collapsed), the server MUST set the **RowNoLongerVisible** field to TRUE, and perform the find from the next row in the table.

RopSetColumns MUST be sent on the table prior to sending RopFindRow. The columns sent for the row found MUST be the columns that are specified on RopSetColumns.

Copyright © 2010 Microsoft Corporation.

Release: Friday, February 5, 2010

The server MUST complete all asynchronous table ROPs prior to executing this ROP or fail the ROP with ecBusy. See section 3.2.5.1.

RopFindRow MUST be supported on contents tables, hierarchy tables, and rules tables.

The following specific error codes apply to this ROP. For more details about ROP errors returned, see Error Codes in [MS-OXCDATA] section 2.4.

Name	Meaning
ecNotSupported	If the object on which this ROP was sent is not a contents, hierarchy, or rules table.

3.2.5.15 Processing RopFreeBookmark

RopFreeBookmark MUST release any resources on the server used to keep track of the bookmark (created using RopCreateBookmark).

If the client does not send RopFreeBookmark, the server MUST release all bookmarks related to a table if that table is released (as a result of sending RopRelease on the table).

RopFreeBookmark MUST be supported by hierarchy tables.

RopFreeBookmark SHOULD be supported by contents tables.<15>

The following specific error codes apply to this ROP. For more details about ROP errors returned, see Error Codes in [MS-OXCDATA] section 2.4.

Name	Meaning
ecNotSupported	If the object on which this ROP was sent is not a hierarchy table.

3.2.5.16 Processing RopResetTable

RopResetTable MUST remove the column set previously specified by RopSetColumns (if any). The columns on the table MUST be treated as if RopSetColumns had never been sent on the table.

RopResetTable MUST remove the restriction previously applied to the table using RopRestrict (if any). The table MUST afterwards appear as if RopRestrict had never been sent on it–as if it had no restriction (all rows MUST be present).

RopResetTable MUST remove the sort order previously applied to the table using RopSortTable (if any). The table MUST afterwards appear as if RopSortTable had never been sent on it–the default sort order is undefined.

RopResetTable MUST clear any errors that currently invalidate the table (if any), such as a failed send to RopSortTable or RopRestrict. Note that even though the errors for the table are cleared, it is not ready to accept RopQueryRows, since its column set has not been specified (the column set can be specified by sending RopSetColumns).

RopResetTable MUST move the cursor to the beginning of the table.

After RopResetTable, all previously existing bookmarks on the table are invalid.

The server MUST complete all asynchronous table ROPs prior to executing this ROP or fail the ROP with ecBusy. See section 3.2.5.1.

Copyright © 2010 Microsoft Corporation.

RopResetTable MUST be supported on contents tables.

The following specific error codes apply to this ROP. For more details about ROP errors returned, see Error Codes in [MS-OXCDATA] section 2.4.

Name	Meaning
ecNotSupported	If the object on which this ROP was sent is not a contents table.

3.2.5.17 Processing RopExpandRow

RopExpandRow MUST set a category row to expanded state.

RopExpandRow MUST be supported for contents tables.

The following specific error codes apply to this ROP. For more details about ROP errors returned, see Error Codes in [MS-OXCDATA] section 2.4.

Name	Meaning
ecNotFound	The row specified by the **CategoryId** field was not found.
ecNotCollapased	The row specified by the **CategoryId** field was not collapsed.
ecNotSupported	If the object on which this ROP was sent is not a contents table.

3.2.5.18 Processing RopCollapseRow

RopCollapseRow MUST set a category row to collapsed state.

RopCollapseRow MUST be supported for contents tables.

The following specific error codes apply to this ROP. For more details about ROP errors returned, see Error Codes in [MS-OXCDATA] section 2.4.

Name	Meaning
ecNotFound	The row specified by the **CategoryId** field was not found.
ecNotExpanded	The row specified by the **CategoryId** field was not expanded.
ecNotSupported	If the object on which this ROP was sent is not a contents table.

3.2.5.19 Processing RopGetCollapseState

RopGetCollapseState MUST send the collapsed state of the whole table in the **CollapseState** field of the response. The collapsed state indicates what categories are expanded. It MUST also include a bookmark to the row indicated by **RowId** and **RowInstanceNumber**.

RopGetCollapseState MUST be supported for contents tables.

The following specific error codes apply to this ROP. For more details about ROP errors returned, see Error Codes in [MS-OXCDATA] section 2.4.

Copyright © 2010 Microsoft Corporation.

Name	Meaning
ecNotSupported	If the object on which this ROP was sent is not a contents table.

3.2.5.20 Processing RopSetCollapseState

RopSetCollapseState MUST modify the collapsed state of the table to match the collapsed state being sent. The collapsed state indicates what categories are expanded. It MUST also move the cursor position to the row specified by the bookmark.

RopSetCollapseState MUST be supported for contents tables.

The following specific error codes apply to this ROP. For more details about ROP errors returned, see Error Codes in [MS-OXCDATA] section 2.4.

Name	Meaning
ecNotSupported	If the object on which this ROP was sent is not a contents table.

3.2.6 Timer Events

There are no timers specific to this protocol.

3.2.7 Other Local Events

There are no other local events specific to this protocol.

Copyright © 2010 Microsoft Corporation.

4 Protocol Examples

The following examples illustrate the byte order of ROPs in a buffer being prepared for transmission. Please note that the examples listed here only show the relevant portions of the specified ROPs; this is not the final byte sequence which gets transmitted over the wire. Also note that the data for a multi-byte field appear in **little-endian** format, with the **BYTES** in the field presented from least significant to most significant. Generally speaking, these ROP requests are packed with other ROP requests and are compressed and packed in one or more **RPC** calls according to the specification in [MS-OXCRPC]. These examples assume the client has already successfully logged on to the server and opened the table. Unless otherwise noted, these examples are additive; the second example is performed after the first example, and so on. For details, see [MS-OXCROPS].

Examples in this section use the following format for byte sequences:

0080: 45 4d 53 4d 44 42 2e 44-4c 4c 00 00 00 00 00 00

The bold value at the far left is the offset of the following bytes into the buffer, expressed in hexadecimal notation. Following the offset is a series of up to 16 **BYTES**, with each two character sequence describing the value of 1 byte in hexadecimal notation. Here, the underlines byte "4d" (01001101) is located 0x83 bytes (131 bytes) from the beginning of the buffer. The dash between eighth byte ("44") and ninth byte ("4c") has no semantic value, and serves only to distinguish the 8-byte boundary for readability purposes.

Such a **BYTE** sequence is then followed by one or more lines interpreting it. In larger examples, the **BYTE** sequence is shown once in its entirety and then repeated in smaller chunks, with each smaller chunk interpreted separately.

The following example shows how a property tag and its property value are represented in a buffer and interpreted directly from it (according to the property Buffer format specified in [MS-OXCDATA]). The property tag appears in the buffer in little-endian format.

0021: 03 00 76 66 0a 00 00-00

PropertyTag: 0x66760003 (PidTagRuleSequence)

PropertyValue: 10

Generally speaking, interpreted values will be shown in their native format, interpreted appropriately from the raw byte sequence as described in the appropriate section. Here, the byte sequence "0a 00 00 00" has been interpreted as a **PtypInteger32** with a value of 10 because the type of the PidTagRuleSequence property is **PtypInteger32**.

4.1 Obtaining a Message List

The following example describes the contents of the ROP request buffer and ROP response buffer for a successful RopGetContentsTable operation, as specified in [MS-OXCFOLD].

4.1.1 Client Request Buffer

A complete ROP request buffer is a 5-byte sequence formatted as follows:

```
0000: 05 00 00 01 00
```

The first 4 bytes are the RopId, **LogonID**, **InputHandleIndex**, and **OutputHandleIndex** fields, as specified in [MS-OXCROPS].

Copyright © 2010 Microsoft Corporation.

Release: Friday, February 5, 2010

```
0000: 05 00 00 01
```

RopId: 0x05 (RopGetContentsTable)

LogonID: 0x00

InputHandleIndex: 0x00. The object for which to obtain the contents table (such as a **Folder Object**).

OutputHandleIndex: 0x01. The location to store the table.

The last byte is the **TableFlags** field, which holds the table Operation flags (detailed in [MS-OXCFOLD] section 2.2.14 RopGetContentsTable).

```
0004: 00
```

TableFlags: 0x00 (Standard)

4.1.2 Server Response to Client Request

```
0000: 05 01 00 00 00 00 04 00-00 00
```

The first 6 bytes of the response buffer are the RopId, **InputHandleIndex**, and **ReturnValue** fields, as specified in [MS-OXCFOLD].

```
0000: 05 01 00 00 00 00
```

RopId: 0x05 (RopGetContentsTable)

InputHandleIndex: 0x01

ReturnValue: 0x00000000 (ecNone: Success)

The next 4 bytes are the **RowCount** field, as specified in [MS-OXCFOLD], which gives the number of rows in the content table.

RowCount: 0x00000004 (four rows in the table)

4.2 Setting the Columns on a Table

The following example describes the contents of the ROP request and response buffers for a successful RopSetColumns operation as specified in section 2.2.2.2.

4.2.1 Client Request Buffer

A complete ROP request buffer is a variable length sequence, with 6 required bytes and 4 bytes for each property tag to be included in the columns set. An example of the request buffer is as follows:

```
0000: 12 00 01 00 06 00 14 00-48 67 14 00 4a 67 14 00
```

Copyright © 2010 Microsoft Corporation.

```
0010: 4d 67 03 00 4e 67 1f 00-37 00 40 00 06 0e
```

The first 3 bytes of the buffer are the **RopId**, LogonID, and **InputHandleIndex** fields of RopSetColumns, as specified in [MS-OXCROPS].

```
0000: 12 00 01
```

RopId: 0x12 (RopSetColumns)

LogonID: 0x00

InputHandleIndex: 0x01

The next 3 bytes are the **SetColumnsFlags** and **PropertyTagCount** fields of RopSetColumns, defined in section 2.2.2.2.1. For more details on property buffer format, see [MS-OXCDATA].

```
0003: 00 06 00
```

SetColumnsFlags: 0x00. Perform this operation synchronously.

PropertyValueCount: 0x0006. Six 4-byte **PropertyTags** follow.

The remaining bytes are the **PropertyTags** field, which holds an array of 4-byte property tags.

```
0006: 14 00 48 67 14 00 4a 67-14 00 4d 67 03 00 4e 67
0016: 1f 00 37 00 40 00 06 0e
```

PropertyTag: 0x67480014 (PidTagFolderId)

PropertyTag: 0x674a0014 (PidTagMid)

PropertyTag: 0x674d0014 (PidTagInstID)

PropertyTag: 0x674e0003 (PidTagInstanceNum)

PropertyTag: 0x0037001f (PidTagSubject)

PropertyTag: 0x0e060040 (PidTagMessageDeliveryTime)

4.2.2 Server Response to Client Request

```
0000: 12 01 00 00 00 00 00
```

The first 6 bytes of the response buffer are the **RopId**, **InputHandleIndex**, and **ReturnValue** fields.

RopId: 0x12 (RopSetColumns)

InputHandleIndex: 0x01

Copyright © 2010 Microsoft Corporation.

Release: Friday, February 5, 2010

ReturnValue: 0x00000000. (ecNone: Success)

The final byte in the response buffer is the **TableStatus** field, described in section 2.2.2.1.3.

```
0006: 00
```

TableStatus: 0x00. This value is TBLSTAT_COMPLETE, indicating that the operation has been completed.

4.3 Sorting a Table by Time Delivered

The following example describes the contents of the ROP request and response buffers for a successful RopSortTable operation as described in section 2.2.2.3.

4.3.1 Client Request Buffer

A complete ROP request buffer is a variable length sequence, with 10 required bytes and 5 bytes for each sorting flag used. An example of the request buffer is as follows:

```
0000: 13 00 01 00 01 00 00 00-00 00 40 00 06 0e 01
```

The first 3 bytes of the buffer are the *RopId*, LogonID, and **InputHandleIndex** fields of RopSetColumns as specified in [MS-OXCROPS].

```
0000: 13 00 01
```

RopId: 0x13 (RopSortTable)

LogonID: 0x00

InputHandleIndex: 0x01. Index in **handle array** for the table to be sorted.

The next 7 bytes are the *SortTableFlags*, *SortOrderCount*, *CategoryCount*, and *ExpandedCount* fields defined in section 2.2.2.3.1.

```
0003: 00 01 00 00 00 00 00
```

SortTableFlags: 0x00. Perform the operation synchronously.

SortOrderCount: 0x0001. Number of *SortOrder* structures to follow

CategoryCount: 0x0000

ExpandedCount: 0x0000

The remaining bytes are the *SortOrders* field, which contain properties to sort by (where there must be exactly SortOrderCount properties) and a sorting method (defined in section 2.2.2.3.1.5).

Copyright © 2010 Microsoft Corporation.

```
000a: 40 00 06 0e 01
```

PropertyTag: 0x0e060040 (PidTagMessageDeliveryTime)

Order: 0x01 (Flag: TABLE_SORT_DESCEND)

4.3.2 Server Response to Client Request

```
0000: 13 01 00 00 00 00 00
```

The first 6 bytes of the response buffer are the *RopId*, *InputHandleIndex*, and *ReturnValue* fields.

```
0000: 13 01 00 00 00 00
```

RopId: 0x13 (RopSortTable)

InputHandleIndex: 0x01

ReturnValue: 0x00000000 (ecNone: Success)

The final byte in the response buffer is the *TableStatus* field described in section 2.2.2.3.2.1.

```
0006: 00
```

TableStatus: 0x00. This value is TBLSTAT_COMPLETE, indicating that the sorting operation has been completed.

4.4 Querying Rows

The following example describes the contents of the ROP request and response buffers for a successful RopQueryRows operation as described in section 2.2.2.5.

4.4.1 Client Request Buffer

A complete ROP request buffer for RopQueryRows is a 7-byte sequence formatted as follows:

```
0000: 15 00 01 00 01 32 00
```

The first 3 bytes are the *RopId*, LogonID, and *InputHandleIndex* fields as specified in [MS-OXCROPS].

```
0000: 15 00 01
```

RopId: 0x15 (RopQueryRows)

LogonID: 0x00

InputHandleIndex: 0x01. The handle of the table to query.

The final 4 bytes of the request buffer are the *QueryRowsFlags*, *ForwardRead*, and *RowCount* fields described in section 2.2.2.5.1.

```
0003: 00 01 32 00
```

QueryRowsFlags: 0x00. Advance the table cursor.

ForwardRead: 0x01. Read the table forward.

RowCount: 0x0032. Return a maximum of 50 rows.

4.4.2 Server Response to Client Request

```
0000: 15 01 00 00 00 00 02 04-00 01 00 01 00 00 00 00
0010: 00 14 88 00 01 00 00 00-0b 3f 87 47 00 01 00 00
0020: 00 0b 3f 87 47 00 00 00-00 00 00 01 00 00 00 0a
0030: 0f 01 04 80 00 49 00 50-00 4d 00 2e 00 4e 00 6f
0040: 00 74 00 65 00 00 00 00-ff ff ff ff 00 00 00 00
0050: 00 0a 0f 01 04 80 00 00-00 00 00 00 23 00 00 00
0060: 0a 0f 01 04 80 0a 0f 01-04 80 0a 0f 01 04 80 0a
0070: 0f 01 04 80 0a 0f 01 04-80 0a 0f 01 04 80 00 ea
0080: 04 00 00 00 1d 05 03 ea-55 73 c8 01 00 6d 00 79
0090: 00 53 00 75 00 62 00 6a-00 65 00 63 00 74 00 00
00a0: 00 00 52 00 45 00 3a 00-20 00 6d 00 79 00 53 00
00b0: 75 00 62 00 6a 00 65 00-63 00 74 00 00 00 00 41
00c0: 00 75 00 74 00 6f 00 55-00 73 00 65 00 72 00 32
00d0: 00 30 00 30 00 30 00 31-00 00 00 0a 0f 01 04 80
00e0: 0a 0f 01 04 80 00 00 00-00 00 01 00 00 00 00 10
00f0: 00 f6 e9 ad 14 41 50 e6-4d 9f 42 64 6e d0 98 c2
```

The first 6 bytes of the response buffer are the *RopId*, *InputHandleIndex*, and *ReturnValue* fields.

```
0000: 15 01 00 00 00 00
```

RopId: 0x15 (RopQueryRows)

InputHandleIndex: 0x01

ReturnValue: 0x00000000 (ecNone: Success)

The next 3 bytes are the *Origin* and *RowCount* fields described in section 2.2.2.5.2.

```
0006: 02 04 00
```

Origin: 0x02. Corresponds to bookmark BOOKMARK_END.

RowCount: 0x0004. Four row property arrays follow in the response.

The remaining bytes in the response buffer are for the *RowData* array, which consists of a **HasError** field for the row and a **ColumnArray** of properties.

Copyright © 2010 Microsoft Corporation.

Release: Friday, February 5, 2010

```
0009: 01 00 01 00 00 00 00 00-14 88 00 01 00 00 00 0b
0019: 3f 87 47 ...
```

HasError: 0x01 (for the row)

ErrorType: 0x00

FlaggedPropertyValue: 0x8814000000000001. From the RopSetColumns operation, this property is PidTagFolderId (0x67480014) because the order must be maintained.

ErrorType: 0x00

FlaggedPropertyValue: 0x47873f0b00000001. From the RopSetColumns operation, this property is PidTagMid (0x674a0014) because the order must be maintained.

This format continues for the remainder of the column properties and then for the remainder of the rows.

4.5 Working with Categories

The following sections give examples of sorting, expanding a row, and querying messages that have been categorized. These examples are separate and do not follow from the ones above.

4.5.1 Sorting a Table by Category

The following example describes the contents of the ROP request and response buffers for a successful RopSortTable operation for category (ascending) and time (descending) sort as described in section 2.2.2.3.

4.5.1.1 Client Request Buffer

A complete ROP request buffer is a variable length sequence, with 10 required bytes and 5 bytes for each sorting flag used. An example of the request buffer is as follows:

```
0000: 13 00 00 00 02 00 01 00-01 00 1f 30 08 80 00 40
0010: 00 06 0e 01
```

The first 3 bytes of the buffer are the *RopId*, LogonID, and *InputHandleIndex* fields of RopSortTable, as specified in [MS-OXCROPS].

```
0000: 13 00 00
```

RopId: 0x13 (RopSortTable)

LogonID: 0x00

InputHandleIndex: 0x00. Index in handle array for the table to be sorted.

The next 7 bytes are the *SortTableFlags*, *SortOrderCount*, *CategoryCount*, and *ExpandedCount* fields defined in section 2.2.2.3.1.

Copyright © 2010 Microsoft Corporation.

Release: Friday, February 5, 2010

```
0003: 00 02 00 01 00 01 00
```

SortTableFlags: 0x00. Perform the operation synchronously.

SortOrderCount: 0x0002. Number of sort order structures to follow.

CategoryCount: 0x0001. There is one category column.

ExpandedCount: 0x0001. All categories are expanded.

The remaining bytes are the *SortOrders* field, which contain properties to sort by (where there must be exactly *SortOrderCount* properties) and a sorting method (defined in section 2.2.2.3.1.5).

```
000a: 1f 30 08 80 00 40 00 06-0e 01
```

PropertyTag: 0x8008301F (PidTagTcvConstLongOne)

Order: 0x00 (Flag: TABLE_SORT_ASCEND)

PropertyTag: 0x0E060040 (PidTagMessageDeliveryTime)

Order: 0x01 (Flag: TABLE_SORT_DESCEND)

4.5.1.2 Server Response to Client Request

```
0000: 13 00 00 00 00 00 00
```

The first 6 bytes of the response buffer are the *RopId*, *InputHandleIndex*, and *ReturnValue* fields.

```
0000: 13 00 00 00 00 00
```

RopId: 0x13 (RopSortTable)

InputHandleIndex: 0x00

ReturnValue: 0x00000000 (ecNone: Success)

The final byte in the response buffer is the *TableStatus* field described in section 2.2.2.3.2.1.

```
0006: 00
```

TableStatus: 0x00. This value is TBLSTAT_COMPLETE, indicating that the sorting operation has been completed.

4.5.2 Expanding a Category Row

The following example describes the contents of the ROP request and response buffers for a successful RopExpandRow operation as described in section 2.2.2.17.

4.5.2.1 Client Request Buffer

A complete ROP request buffer is a 13-byte sequence, formatted as follows:

```
0000: 59 00 01 00 00 01 00 00-00 00 f1 88 bd
```

The first 3 bytes of the buffer are the *RopId*, LogonID, and *InputHandleIndex* fields of RopExpandRow as specified in [MS-OXCROPS].

```
0000: 59 00 00
```

RopId: 0x59 (RopExpandRow)

LogonID: 0x00

InputHandleIndex: 0x01

The remaining 13 bytes are the *MaxRowCount* and *CategoryId* fields described in section 2.2.2.17.1.

```
0003: 00 00 01 00 00 00 00 f1-88 bd
```

MaxRowCount: 0x0000. rows will be expanded but not returned in the response.

CategoryId: 0xbd88f10000000001. The PidTagInstID of the category row to expand.

4.5.2.2 Server Response to Client Request

```
0000: 59 01 00 00 00 00 03 00-00 00 00 00
```

The first 6 bytes of the response buffer are the *RopId*, *InputHandleIndex*, and *ReturnValue* fields.

```
0000: 59 00 00 00 00 00
```

RopId: 0x59 (RopExpandRow)

InputHandleIndex: 0x00

ReturnValue: 0x00000000 (ecNone: Success)

The remaining bytes are the *ExpandedRowCount*, *RowCount*, and *RowData* fields described in section 2.2.2.17.2.

```
0006: 03 00 00 00 00 00
```

ExpandedRowCount: 0x00000003. There are a total of three rows in the expanded category.

RowCount: 0x0000. No row data follows.

Copyright © 2010 Microsoft Corporation.

RowData: [EMPTY]

4.5.3 Querying Rows with Category View

The following example describes the contents of the ROP request and response buffers for a successful RopQueryRows operation as described in section 2.2.2.17 when the messages are grouped by category.

4.5.3.1 Client Request Buffer

A complete ROP request buffer for RopQueryRows is a 7-byte sequence formatted as follows:

```
0000: 15 00 00 00 01 32 00
```

The first 3 bytes are the *RopId*, LogonID, and *InputHandleIndex* fields as specified in [MS-OXCROPS].

```
0000: 15 00 00
```

RopId: 0x15 (RopQueryRows)

LogonID: 0x00

InputHandleIndex: 0x00. The handle of the table to query.

The final 4 bytes of the request buffer are the QueryRowsFlags, ForwardRead, and RowCount fields described in section 2.2.2.5.2.

```
0003: 00 01 32 00
```

QueryRowsFlags: 0x00. Advance the table cursor.

ForwardRead: 0x01. Read the table forward.

RowCount: 0x0032. Return a maximum of 50 rows.

4.5.3.2 Server Response to Client Request

```
0000: 15 00 00 00 00 00 02 09-00 01 00 01 00 00 00 00
0010: f1 1f 32 0a 0f 01 04 80-00 01 00 00 00 00 f1 88
0020: bd 00 00 00 00 00 00 03-00 00 00 00 00 00 00 00
0030: 0a 0f 01 04 80 0a 0f 01-04 80 0a 0f 01 04 80 0a ...
```

The first 6 bytes of the response buffer are the *RopId*, *InputHandleIndex*, and *ReturnValue* fields.

```
0000: 15 00 00 00 00 00
```

RopId: 0x15 (RopQueryRows)

InputHandleIndex: 0x00

ReturnValue: 0x00000000 (ecNone: Success)

The next 3 bytes are the **Origin** and **RowCount** fields described in section 2.2.2.5.2.

```
0006: 02 09 00
```

Origin: 0x02. Corresponds to bookmark BOOKMARK_END.

RowCount: 0x0009. Nine row property arrays follow in the response.

The remaining bytes in the response buffer are for the **RowData** array, which consists of a **HasError** field for the row and a **ColumnArray** of properties. The RopSetColumns request for this sequence of ROPs has not been shown.

```
0009: 01 00 01 00 00 00 00 f1-1f 32 0a 0f 01 04 80 00
0019: 01 00 00 00 00 f1 88 bd-00 00 00 00 00 00 03 00
0029: 00 00 00 00 00 00 00 0a-0f 01 04 80 0a 0f 01 04
0039: 80 0a 0f 01 04 80 0a ...
```

HasError: 0x01. For the entire row.

ErrorType: 0x00.

FlaggedPropertyValue: 0x321ff10000000001. This property is PidTagFolderId (0x67480014).

The format follows this pattern as covered in 4.4.2, the server response buffer for the first RopQueryRows example.

Because this example is for messages with categories, there is an interesting case when one Message has multiple categories assigned to it. Further into the buffer, there are the following sets of properties:

property tag	**property value**
0x674D0014 (PidTagInstID)	0xb773f10000000001
0x674E0003 (PidTagInstanceNum)	1
0x8008001F (PidTagTcvConstLongOne)	Category1

property tag	**property value**
0x674D0014 (PidTagInstID)	0xb773f10000000001
0x674E0003 (PidTagInstanceNum)	2
0x8008001F (PidTagTcvConstLongOne)	Category2

The same message appears twice in the contents table due to the category grouping. The PidTagInstanceNum property makes this phenomenon easily recognizable.

5 Security

5.1 Security Considerations for Implementers

There are no special security considerations specific to the table Object protocol. General security considerations pertaining to the underlying protocol apply, as specified in [MS-OXCROPS].

5.2 Index of Security Parameters

None.

Copyright © 2010 Microsoft Corporation.

Release: Friday, February 5, 2010

6 Appendix A: Product Behavior

The information in this specification is applicable to the following product versions. References to product versions include released service packs.

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

Exceptions, if any, are noted below. If a service pack number appears with the product version, behavior changed in that service pack. The new behavior also applies to subsequent service packs of the product unless otherwise specified.

Unless otherwise specified, any statement of optional behavior in this specification 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 product does not follow the prescription.

<1> Section 2: Exchange 2010 does not support the following ROPs when client connection services are deployed on an Exchange server that does not also have a **Mailbox** store installed:

RopAbortSubmit

RopBackOff

RopCollapseRow

RopCopyToStream

RopCreateBookmark

RopDeletePropertiesNoReplicate

RopExpandRow

RopFastTransferSourceCopyTo

RopFreeBookmark

RopGerPerUserGuid

RopGetCollapseState

RopGetOwningServers

RopGetPerUserLongTermIds

RopGetReceiveFolderTable

RopGetStatus

RopGetStoreState

RopHardDeleteMessages

RopHardDeleteMessagesAndSubfolders

RopLockRegionStream

RopPending

RopPublicFolderIsGhosted

RopQueryColumnsAll

RopQueryNamedProperties

RopReadPerUserInformation

RopRegisterSynchronizationNotifications

RopSeekRowBookmark

RopSeekRowFractional

RopSeekStream

RopSetCollapseState

RopSetPropertiesNoReplicate

RopSetReceiveFolder

RopSetSynchronizationNotificationGuid

RopSynchronizationOpenAdvisor

RopUnlockRegionStream

RopUpdateDeferredActionMessages

RopWritePerUserInformation

<2> Section 2.2.2.1.2: Exchange 2010 does not support custom bookmarks when client connection services are deployed on an Exchange server that does not also have a Mailbox store installed.

<3> Section 2.2.2.2.1.3: Exchange 2010 does not support this ROP when client connection services are deployed on an Exchange server that does not also have a Mailbox store installed.

<4> Section 2.2.2.10.1.2: Exchange 2010 does not support this ROP when client connection services are deployed on an Exchange server that does not also have a Mailbox store installed.

<5> Section 2.2.2.18: Exchange 2010 does not support this ROP when client connection services are deployed on an Exchange server that does not also have a Mailbox store installed.

<6> Section 2.2.2.19: Exchange 2010 does not support this ROP when client connection services are deployed on an Exchange server that does not also have a Mailbox store installed.

<7> Section 2.2.2.19.2.2: Exchange 2010 does not support this ROP when client connection services are deployed on an Exchange server that does not also have a Mailbox store installed.

Copyright © 2010 Microsoft Corporation.

<8> Section 3.1.4.1.1: Outlook 2007 never performs asynchronous table ROPs against the server.

<9> Section 3.1.4.1.1: Exchange 2010 can output unexpected results for RopAbort when client connection services are deployed on an Exchange server that does not also have a mailbox store installed.

<10> Section 3.1.4.1.1: Exchange 2010 does not support this ROP when client connection services are deployed on an Exchange server that does not also have a mailbox store installed.

<11> Section 3.1.4.2: Exchange 2010 does not support this ROP when client connection services are deployed on an Exchange server that does not also have a Mailbox store installed.

<12> Section 3.1.4.3: Exchange 2010 does not support this ROP when client connection services are deployed on an Exchange server that does not also have a Mailbox store installed.

<13> Section 3.1.4.3: Exchange 2010 does not support this ROP when client connection services are deployed on an Exchange server that does not also have a Mailbox store installed.

<14> Section 3.1.4.3: Exchange 2010 does not support this ROP when client connection services are deployed on an Exchange server that does not also have a Mailbox store installed.

<15> Section 3.2.5.15: Exchange 2003 and Exchange 2007 do not conform to the specification and currently send the value ecNotSupported in the ReturnValue field of the ROP response buffer for contents tables.

7 Change Tracking

This section identifies changes made to [MS-OXCTABL] protocol documentation between November 2009 and February 2010 releases. Changes are classed as major, minor, or editorial.

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.
- A protocol is deprecated.
- The removal of a document from the documentation set.
- Changes made for template compliance.

Minor changes do not affect protocol interoperability or implementation. Examples are updates to fix technical accuracy or ambiguity at the sentence, paragraph, or table level.

Editorial changes apply to grammatical, formatting, and style issues.

No changes means that the document is identical to its last release.

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

- New content added.
- Content update.
- 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.

Copyright © 2010 Microsoft Corporation.

- Content removed for template compliance.
- Obsolete document removed.

Editorial changes always have the revision type "Editorially updated."

Some important terms used in revision 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.

Changes are listed in the following table. If you need further information, please contact protocol@microsoft.com.

Section	**Tracking number (if applicable) and description**	**Major change (Y or N)**	**Revision Type**
3.2.5.5 Processing RopQueryRows	50220 Provided a reference to [MS-OXCDATA] for ecBufferTooSmall.	N	Content update.

8 Index

Copyright © 2010 Microsoft Corporation.

Release: Friday, February 5, 2010