[MS-OXOTASK]:
Task-Related Objects Protocol Specification

Intellectual Property Rights Notice for Open Specifications Documentation

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		Updated IP notice.
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	4.0.0	Major	Updated and revised the technical content.
02/10/2010	5.0.0	Major	Updated and revised the technical content.
05/05/2010	5.0.1	Editorial	Revised and edited the technical content.

Table of Contents

Release: Saturday, May 1, 2010

1 Introduction

This document specifies the Task-Related Objects Protocol, which defines several objects that model the electronic equivalent of tasks, task assignments, and **task updates**. These objects maintain basic task information, such as a description, notes, due date, **reminder** time, status, assignment acceptance, and more.

1.1 Glossary

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

    Attachment object
    Bcc recipient
    Cc recipient
    contents table
    Coordinated Universal Time (UTC)
    delegate
    EntryID
    Folder object
    GUID
    handle
    Long ID (LID)
    mailbox
    Message object
    named property
    primary recipient
    Property (1)
    property ID
    property name
    public folder
    recipient (2)
    recurrence pattern
    recurring task
    reminder
    remote operation (ROP)
    Simple Mail Transfer Protocol (SMTP)
    special folder
    tagged property
    Task object
    task request
    Unicode

The following terms are specific to this document:

display name: A label used to identify an object to the user.

task acceptance: A **Message object** that is used to convey acceptance of a task assignment.

task assignee: A user to whom a task has been assigned.

task assigner: A user who assigns a task to another user.

task communication: Collectively, a **task request**, a **task acceptance** or **task rejection**, and a **task update**.

task owner: The user who is responsible for updating a task. For unassigned tasks, the local user is the owner; the **task assignee** is the owner of assigned tasks.

task rejection: A **Message object** that is used to convey rejection of a task assignment.

task response: A **task acceptance** or a **task rejection**.

task update: A **Message object** that is used by a **task assignee** to send task changes to a **task assigner**.

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

1.2   References

1.2.1   Normative References

We conduct frequent surveys of the normative references to assure their continued availability. If you have any issue with finding a normative reference, please contact dochelp@microsoft.com. We will assist you in finding the relevant information. Please check the archive site, http://msdn2.microsoft.com/en-us/library/E4BD6494-06AD-4aed-9823-445E921C9624, as an additional source.

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

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

[MS-OXCPRPT] Microsoft Corporation, "Property and Stream Object Protocol Specification", April 2008.

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

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

[MS-OXOCAL] Microsoft Corporation, "Appointment and Meeting Object Protocol Specification", April 2008.

[MS-OXODLGT] Microsoft Corporation, "Delegate Access Configuration Protocol Specification", April 2008.

[MS-OXOMSG] Microsoft Corporation, "E-Mail Object Protocol Specification", April 2008.

[MS-OXORMDR] Microsoft Corporation, "Reminder Settings Protocol Specification", April 2008.

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

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

1.2.2   Informative References

None.

1.3 Overview

The Task-Related Objects Protocol provides an electronic mechanism for tracking tasks, to-do items, and assignments.

The Task-Related Objects Protocol allows for the representation of task-related **Message objects** in a messaging store. It extends the Message and Attachment Object protocol in that it defines new **properties** and adds restrictions to the properties that the protocol defines. For details about the Message and Attachment Object protocol, see [MS-OXCMSG].

The task representation is a **Task object**. The properties that are specific to a Task object facilitate retaining information about the due date, assignment status, and anticipated work effort, among other things, of the task. A Task object is stored in a **Folder Object**. The Task-Related Objects Protocol specifies how a Task object is created and manipulated. It also specifies how task assignments are made, confirmed, and updated through the use of **task communications**, which include **task requests**, **task acceptances**, **task rejections**, and task updates. The Task-Related Objects Protocol also specifies how a series of tasks can be generated from a single Task object with a **recurrence pattern**.

1.4 Relationship to Other Protocols

The Task-Related Objects Protocol has the same dependencies as the Message and **Attachment Object** protocol, which it extends. For details about the Message and Attachment Object protocol, see [MS-OXCMSG].

The Task-Related Objects Protocol is a peer of the E-mail Object protocol, and uses a subset of the properties and **ROPs** specified by the E-mail Object protocol. For details about the E-mail Object protocol, see [MS-OXOMSG].

1.5 Prerequisites/Preconditions

The Task-Related Objects Protocol has the same prerequisites and preconditions as the Message and Attachment Object protocol. For details about the Message and Attachment Object protocol, see [MS-OXCMSG].

1.6 Applicability Statement

The Task-Related Objects protocol is appropriate for clients and servers that manage user tasks and their associated resources.

1.7 Versioning and Capability Negotiation

None.

1.8 Vendor-Extensible Fields

This protocol provides no vendor extensibility beyond what is already specified in [MS-OXCMSG].

1.9 Standards Assignments

None.

2   Messages

2.1   Transport

The Task-Related Objects Protocol uses the Property and Stream Object protocol, as specified in [MS-OXCPRPT], and the Message and Attachment Object protocol, as specified in [MS-OXCMSG], as its primary transport mechanism.

2.2   Message Syntax

A Task object and a task communication can be created and modified by clients and servers. Except where noted, this section defines constraints under which both clients and servers operate.

Clients operate on a Task object and task communication by using the Message and Attachment Object protocol, as specified in [MS-OXCMSG]. How a server operates on a task object and a task communication is implementation-dependent, but the results of any such operations MUST be exposed to clients in a manner that is consistent with the Task-Related Objects protocol.

Unless otherwise specified, a Task object and a task communication MUST adhere to all property constraints specified in [MS-OXPROPS], [MS-OXCMSG], and [MS-OXOMSG]. A Task object and a task communication can contain additional properties <1><2><3>, which are specified in [MS-OXPROPS], but these additional properties have no impact on the Task-Related Objects protocol.

The property data types are specified in [MS-OXCDATA] section 2.12.1.

2.2.1   Folder Properties

Properties in this section are set on a Folder object in which task objects are stored.

2.2.1.1   PidTagOrdinalMost

Type: **PtypInteger32**

Contains a positive number whose negative is less than or equal to the value of PidLidTaskOrdinal of all Task objects in the folder. This property MUST be updated to maintain this condition whenever the PidLidTaskOrdinal property of any Task object in the folder changes in a way that would violate the condition.

2.2.2   Task Object Properties

This section specifies property requirements for Task objects.

2.2.2.1   Additional Property Constraints

In some cases, the Task object has specific requirements for properties that are otherwise inherited. This section specifies these specific requirements.

2.2.2.1.1   PidTagMessageClass

Type: **PtypString8**, case-insensitive

Specifies the type of the Message object. The value MUST be "`IPM.Task`" or begin with "`IPM.Task.`".

2.2.2.1.2   Body properties

The specifications in [MS-OXCMSG] regarding Rich Text body properties apply to Task objects.

2.2.2.1.3   PidLidCommonStart

Type: **PtypTime**, **UTC**

This value MUST be the UTC equivalent of the PidLidTaskStartDate property.

2.2.2.1.4   PidLidCommonEnd

Type: **PtypTime**, UTC

This value MUST be the UTC equivalent of the PidLidTaskDueDate property.

2.2.2.1.5   PidTagIconIndex

Type: **PtypInteger32**

Specifies which icon is to be used by a user interface to represent the Task object. If this property exists, its value is a hint to the client's user interface. The user interface can ignore the value and use another method of determining which icon to display to the user.

The value is one of the following:

Value	Meaning
0x00000501	The Task object has not been assigned and is a **recurring task**.
0x00000502	The Task object is the **task assignee's** copy of the Task object.
0x00000503	The Task object is the **task assigner's** copy of the Task object.
0x00000500	None of the other conditions apply.

2.2.2.2   Task Object Specific Properties

2.2.2.2.1   PidLidTaskMode

Type: **PtypInteger32**

Specifies the assignment status of the Task object. The value is one of the following:

Value	Meaning
0x00000000	The Task object is not assigned.
0x00000001	The Task object is embedded in a task request.
0x00000002	The Task object has been accepted by the task assignee.
0x00000003	The Task object was rejected by the task assignee.
0x00000004	The Task object is embedded in a task update.

Value	Meaning
0x00000005	The Task object was assigned to the task assigner (self-delegation).

2.2.2.2.2 PidLidTaskStatus

Type: **PtypInteger32**

Specifies the status of the user's progress on the task. The value is one of the following:

Value	Meaning
0x00000000	The user has not started work on the Task object. If this value is set, PidLidPercentComplete MUST be 0.0.
0x00000001	The user's work on this Task object is in progress. If this value is set, PidLidPercentComplete MUST be greater than 0.0 and less than 1.0.
0x00000002	The user's work on this Task object is complete. If this value is set, PidLidPercentComplete MUST be 1.0, PidLidTaskDateCompleted MUST be the current date, and PidLidTaskComplete MUST be 0x01.
0x00000003	The user is waiting on somebody else.
0x00000004	The user has deferred work on the Task object.

2.2.2.2.3 PidLidPercentComplete

Type: **PtypFloating64**

Indicates the progress the user has made on a task. The value MUST be a number greater than or equal to 0.0 and less than or equal to 1.0, where 1.0 indicates that work is completed and 0.0 indicates that work has not begun.

2.2.2.2.4 PidLidTaskStartDate

Type: **PtypTime**, in the user's local time zone.

The date on which the user expects work on the task to begin. If left unset, the task does not have a start date. A value of 0x5AE980E0 (1,525,252,320) also means that the task does not have a start date. If the task has a start date, the value MUST have a time component of 12:00 midnight, and PidLidTaskDueDate and PidLidCommonStart MUST also be set.

2.2.2.2.5 PidLidTaskDueDate

Type: **PtypTime**, in the user's local time zone.

The date by which the user expects work on the task to be complete. The task has no due date if this property is unset or set to 0x5AE980E0 (1,525,252,320). However, a due date is optional only if no start date is indicated in PidLidTaskStartDate. If the task has a due date, the value MUST have a time component of 12:00 midnight, and PidLidCommonEnd MUST also be set. If PidLidTaskStartDate has a start date, then the value of this property MUST be greater than or equal to the value of PidLidTaskStartDate.

2.2.2.2.6 PidLidTaskResetReminder

Type: **PtypBoolean**

Indicates whether future instances of recurring tasks need reminders, even though PidLidReminderSet is FALSE (0x00). The PidLidTaskResetReminder property is set to TRUE (0x01) when the task's reminder is dismissed, and set to FALSE (0x00) otherwise. If left unset, a default of FALSE (0x00) is assumed.

As specified in [MS-OXORMDR], the PidLidReminderSet property indicates whether a reminder is set on the Task object. However, this property only indicates the presence of a reminder on a single Task object. It cannot be used alone to determine whether a future instance of a recurring task needs a reminder.

This is best understood by example. Suppose that the user wants reminders for a series of recurring tasks. The client creates a Task object and sets PidLidReminderSet to TRUE (0x01). At the appropriate time, the client presents the user with a reminder. When the user dismisses the reminder, the client sets PidLidReminderSet to FALSE (0x00) and sets PidLidTaskResetReminder to TRUE (0x01). Later, the user completes the task and the client generates a new occurrence of the Task object. As stated, the user wanted the new occurrence to have a reminder, but the last known value of PidLidReminderSet was FALSE (0x00). The client uses the TRUE (0x01) value of PidLidTaskResetReminder to decide that the user had set and dismissed a reminder on a previous occurrence of the task. If the value had been FALSE (0x00), the client would decide that the user had never set a reminder on the task at all. The client sets a new reminder, as specified in [MS-OXORMDR], if either PidLidReminderSet or PidLidTaskResetReminder is TRUE (0x01).

2.2.2.2.7 PidLidTaskAccepted

Type: **PtypBoolean**

Indicates whether a task assignee has replied to a task request for this Task object. The client sets this property to FALSE (0x00) for a new Task object and to TRUE (0x01) when a Task object is either accepted or rejected. If left unset, a value of FALSE (0x00) is assumed.

2.2.2.2.8 PidLidTaskDeadOccurrence

Type: **PtypBoolean**

Indicates whether new occurrences remain to be generated.

A recurrence pattern is no longer in effect when its final instance is in the past or its specified number of instances has been generated.

The client sets this property to FALSE (0x00) for a new Task object and to TRUE (0x01) when it generates the last instance of a recurring task. Also, when copying a Task object as part of generating a new instance, this property is set to TRUE (0x01) on the copy (which is the completed instance).

2.2.2.2.9 PidLidTaskDateCompleted

Type: **PtypTime**, in UTC.

The date when the user completed work on the task. This property can be left unset; if set, this property MUST have a time component of 12:00 midnight in the local time zone, converted to UTC.

2.2.2.2.10   PidLidTaskLastUpdate

Type: **PtypTime**, in UTC.

The date and time of the most recent change made to the Task object (indicated by the PidLidTaskHistory property).

2.2.2.2.11   PidLidTaskActualEffort

Type: **PtypInteger32**

Indicates the number of minutes that the user actually spent working on a task. The value MUST be greater than or equal to zero and less than 0x5AE980DF (1,525,252,319), where 480 minutes equal one day and 2400 minutes equal one week (8 hours in a work day and 5 work days in a work week).

2.2.2.2.12   PidLidTaskEstimatedEffort

Type: **PtypInteger32**

Indicates the number of minutes that the user expects to work on a task. The value MUST be greater than or equal to zero and less than 0x5AE980DF (1,525,252,319), where 480 minutes equal one day and 2400 minutes equal one week (8 hours in a work day and 5 work days in a work week).

2.2.2.2.13   PidLidTaskVersion

Type: **PtypInteger32**

Indicates which copy is the latest update of a Task object. An update with a lower version than the Task object is ignored. When embedding a Task object in a task communication, the client sets the current version of the embedded Task object on the task communication as well.

The initial value of this property is 1. The value is incremented only when the Task object is owned by the user. The Task object is owned by the user when PidLidTaskState is set to 0x00000001, 0x00000002, or 0x00000004.

2.2.2.2.14   PidLidTaskState

Type: **PtypInteger32**

Indicates the current assignment state of the Task object. The value is one of the following:

Value	Meaning
0x00000001	The Task object is not assigned.
0x00000002	The Task object is the task assignee's copy of an assigned Task object.
0x00000003	The Task object is the task assigner's copy of an assigned Task object.
0x00000004	The Task object is the task assigner's copy of a rejected Task object.
0x00000000	This Task object was created to correspond to a Task object that was embedded in a task rejection but could not be found locally.

2.2.2.2.15   PidLidTaskRecurrence

Type: **PtypBinary**

Contains a **RecurrencePattern** structure that provides information about recurring tasks. Both the **DeletedInstanceCount** field and the **ModifiedInstanceCount** field of the **RecurrencePattern** structure MUST be set zero. For details about the format of the **RecurrencePattern** structure, see [MS-OXOCAL] section 2.2.1.44.1.

2.2.2.2.16   PidLidTaskAssigners

Type: **PtypBinary**

Contains a stack of entries, each representing a task assigner. The most recent task assigner (that is, the top of the stack) appears at the end.

Size in bytes	Type	Name	Notes
4	**PtypInteger32**	cAssigners	Number of Task assigners.
4	**PtypInteger32**	cbAssigner	Size of the Task assigner data to follow, in BYTEs.
Variable	**Address Book EntryID** structure (for more details about this structure, see [MS-OXCDATA] section 2.2.5.2)	EID	Task assigner's Address Book **EntryID**.
Variable	**PtypString8**	szDisplayName	Task assigner's display name, using multi-byte characters.
Variable	**PtypString**	wzDisplayName	Task assigner's display name, using **Unicode** characters.
Next task assigner's data begins here.			

When the client receives a task request, it appends to this property an entry representing the sender of the task request, pursuant to the structure specified above.

When the client receives a task rejection, the client removes the last task assigner entry from this property, pursuant to the structure specified above.

When the client sends a **task response**, the client sends it to the last task assigner listed in the value of this property.<4>

2.2.2.2.17   PidLidTaskStatusOnComplete

Type: **PtypBoolean**

Indicates whether the task assignee has been requested to send an e-mail message update when the task assignee completes the assigned task. The client sets this property to TRUE (0x01) when the task assignee has been requested to send an e-mail message update; otherwise, this property is set to FALSE (0x00).

2.2.2.2.18   PidLidTaskHistory

Type: **PtypInteger32**

Indicates the type of change that was last made to the Task object. When the value of this property is set, the PidLidTaskLastUpdate property MUST also be set to the current time.

The value is one of the following (listed in order of decreasing priority):

Value	Meaning
0x00000004	The PidLidTaskDueDate property changed.
0x00000003	Another property was changed.
0x00000001	The task assignee accepted this Task object.
0x00000002	The task assignee rejected this Task object.
0x00000005	The Task object has been assigned to a task assignee.
0x00000000	No changes were made.

2.2.2.2.19   PidLidTaskUpdates

Type: **PtypBoolean**

Indicates whether the task assignee has been requested to send a task update when the assigned Task object changes. The client sets this property to TRUE (0x01) when the task assignee has been requested to send a task update; otherwise, this property is set to FALSE (0x00).

2.2.2.2.20   PidLidTaskComplete

Type: **PtypBoolean**

Indicates whether the task has been completed. The client sets this property to TRUE (0x01) when the task has been completed; otherwise, this property is set to FALSE (0x00).

2.2.2.2.21   PidLidTaskFCreator

Type: **PtypBoolean**

Indicates that the Task object was originally created by the action of the current user or user agent instead of by the processing of a task request. The client sets this property to TRUE (0x01) when the user creates the task and to FALSE (0x00) when the task was assigned by another user. If left unset, a value of TRUE (0x01) is assumed.

2.2.2.2.22   PidLidTaskOwner

Type: **PtypString**

The name of the **task owner**.

2.2.2.2.23   PidLidTaskMultipleRecipients

Type: **PtypInteger32**

Provides optimization hints about the recipients of a Task object.

This property can be left unset; if set, it MUST be set to a bitwise OR of zero or more of the following values:

Name	Value	Meaning
Sent	0x00000001	The Task object has multiple **primary recipients**.
Received	0x00000002	Although the "Sent" hint was not present, the client detected that the Task object has multiple primary recipients.
Reserved	0x00000004	This value is reserved<5>.

2.2.2.2.24   PidLidTaskAssigner

Type: P**typString**

The name of the user that last assigned the task. Left unset if the task has not been assigned.

Because this property is set by the client after the task assignee receives a task request, the property will not be set on the task assigner's copy of the Task object.

When the client adds or removes a task assigner from the stack of task assigners listed in the PidLidTaskAssigners property (for details, see 2.2.2.2.16), this property is set to the added or removed task assigner.

2.2.2.2.25   PidLidTaskLastUser

Type: **PtypString**

The name of the most recent user to have been the task owner.

Before a client sends a task request, it sets this property to the name of the task assigner.

Before a client sends a task acceptance, it sets this property to the name of the task assignee.

Before a client sends a task rejection, it sets this property to the name of the task assigner.

2.2.2.2.26   PidLidTaskOrdinal

Type: **PtypInteger32**

An aid to custom sorting of Task objects. This property can be left unset; if set, its value MUST be greater than 0x800186A0 (-2,147,383,648) and less than 0x7FFE7960 (2,147,383,648) and MUST be unique among Task objects in the same folder.

Whenever the client sets this property to a number less than the negative of the current value of the PidTagOrdinalMost property of the folder, the client MUST also update PidTagOrdinalMost on the folder.

The PidTagOrdinalMost property of the folder provides an efficient way to determine a unique value among Task objects in the same folder.

2.2.2.2.27   PidLidTaskLastDelegate

Type: **PtypString**

The name of the **mailbox's delegate** who most recently assigned the task, or an empty string if there is no delegate. For details about delegates, see [MS-OXODLGT].

2.2.2.2.28   PidLidTaskFRecurring

Type: **PtypBoolean**

Indicates whether the task includes a recurrence pattern. If left unset, a default value of FALSE (0x00) is assumed. If set to TRUE (0x01), the PidLidTaskRecurrence and PidLidTaskDeadOccurrence properties MUST also be set, as specified in sections 2.2.2.2.15 and 2.2.2.2.8, respectively.

2.2.2.2.29   PidLidTaskOwnership

Type: **PtypInteger32**

Indicates the role of the current user relative to the Task object. The value is one of the following:

Value	Meaning
0x00000000	The Task object is not assigned.
0x00000001	The Task object is the task assigner's copy of the Task object.
0x00000002	The Task object is the task assignee's copy of the Task object.

2.2.2.2.30   PidLidTaskAcceptanceState

Type: **PtypInteger32**

Indicates the acceptance state of the task. The value is one of the following:

Value	Meaning
0x00000000	The Task object is not assigned.
0x00000001	The Task object's acceptance status is unknown.
0x00000002	The task assignee has accepted the Task object. This value is set when the client processes a task acceptance.
0x00000003	The task assignee has rejected the Task object. This value is set when the client processes a task rejection.

2.2.2.2.31   PidLidTaskFFixOffline

Type: **PtypBoolean**

Indicates whether the value of the PidLidTaskOwner property is correct. The value is one of the following.

Value	Meaning
FALSE (0x00) or unset	The value for PidLidTaskOwner is correct.
TRUE (0x01)	The client cannot determine an accurate value for PidLidTaskOwner.

When setting a value of TRUE (0x01), the client can also set the PidLidTaskOwner property to a generic owner name, such as "Unknown".

If the PidLidTaskFFixOffline property is set to TRUE (0x01) and the client can determine the owner, then the client can update the PidLidTaskOwner property and set the PidLidTaskFFixOffline property to FALSE (0x00).

2.2.2.2.32 PidLidTaskGlobalId

Type: **PtypBinary**

A unique **GUID** for this task, used to locate an existing task upon receipt of a task response or task update. This property MUST be set for assigned tasks, but it can be left unset for unassigned tasks.

2.2.3 Task Communications Properties

This section specifies property requirements that are specific to task requests, task acceptances, task rejections, and task updates (collectively, task communications).

2.2.3.1 PidTagProcessed

Type: **PtypBoolean**

Indicates whether a client has already processed a received task communication. Left unset until processing has completed, then set to TRUE (0x01).

2.2.3.2 PidLidTaskMode

Type: **PtypInteger32**

Specifies the assignment status of the embedded Task object. The value MUST be the same value that is stored in the PidLidTaskMode property of the embedded Task object.

2.2.3.3 Additional Property Constraints

In some cases, the task communication has specific requirements for properties that are otherwise inherited. This section specifies these specific requirements.

2.2.3.3.1 PidTagMessageClass

Type: **PtypString8**, case-insensitive.

Specifies the type of the Message object. The value is one of the following strings:

String	Type of task communication
"IPM.TaskRequest"	Task request
"IPM.TaskRequest.Accept"	Task acceptance
"IPM.TaskRequest.Decline"	Task rejection
"IPM.TaskRequest.Update"	Task update

2.2.3.3.2 PidTagIconIndex

Type: **PtypInteger32**

Specifies which icon is to be used by a user interface to represent the task communication. If this property exists, its value is a hint to the client's user interface. The user interface can ignore the value and use another method of determining which icon to display to the user.

The value is one of the following:

Value	Type of task communication
0x00000504	Task request
0x00000505	Task acceptance
0x00000506	Task rejection
0x00000500	Task update
0xFFFFFFFF	Unspecified

3 Protocol Details

General protocol details, as specified in [MS-OXPROPS] and [MS-OXCMSG], apply, unless otherwise specified in the following sections.

3.1 Client Details

3.1.1 Abstract Data Model

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

3.1.1.1 Task Objects and Task Communications

Task objects and task communications extend the Message object. For details about Message objects, see [MS-OXCMSG].

3.1.1.2 Folder Objects for Task Objects

A Task object is created in the Tasks **special folder**, as specified in [MS-OXOSFLD], unless the end user or user agent explicitly specifies another Folder object.

3.1.2 Timers

None.

3.1.3 Initialization

None.

3.1.4 Higher-Layer Triggered Events

3.1.4.1 Creation of Task Objects and Task Communications

To create task objects and task communications, the client or server creates a Message object as specified in [MS-OXCMSG], sets properties in accordance with the requirements both in 2 of this **document** and in [MS-OXCPRPT], and saves and/or submits the resulting Message objects as specified in [MS-OXCMSG] and [MS-OXOMSG].

3.1.4.2 Modification of Task Objects and Task Communications

When modifying Task objects and task communications, the client or server opens a Message object as specified in [MS-OXCMSG], modifies any of the properties in accordance with the requirements both in 2 of this document and in [MS-OXCPRPT], and saves the Message object as specified in [MS-OXCMSG].

3.1.4.3 Embedding Task Objects

A task communication conveys a request or response about a Task object. To identify the Task object, the client embeds a copy of the Task object as an Attachment object within the task communication (the embedding object).

To embed a Task object, the client MUST complete the following steps in the order specified:

1. Create an Attachment object on the embedding object, as specified in [MS-OXCMSG]. This attachment MUST be the first Attachment object created on the embedding object.

2. Set PidTagAttachMethod to afEmbeddedMessage (0x00000005), PidTagRenderingPosition to 0xFFFFFFFF, and PidTagAttachmentHidden to 0x01, as specified in [MS-OXCMSG]<6>.

3. Open the Attachment object as an embedded Message object, as specified in [MS-OXCMSG].

4. Set the appropriate properties of the embedded Message object (the embedded Task object) as specified throughout this document.

5. If the original Task object has a PidLidTaskGlobalId property, copy it to the embedded Task object. Otherwise, set the value of the PidLidTaskGlobalId property of the embedded Task object to a new, unique GUID.

6. Save the embedded Message object, as specified in [MS-OXCMSG].

7. Save the Attachment object, as specified in [MS-OXCMSG].

8. Release the Message object, as specified in [MS-OXCMSG].

9. Release the Attachment object, as specified in [MS-OXCMSG].

3.1.4.4 Creating Task Objects and Task Communications

Task objects and task communications are all created the same way. They differ in the properties and property values that are set on them.

To create a Task object or task communication, the client creates a Message object, as specified in [MS-OXCMSG], and sets the type-specific properties, as specified in section 2 of this document. To send task communications, the client addresses them to the appropriate recipients, as specified in [MS-OXOMSG], and submits the Message object, as specified in [MS-OXOMSG]. When creating task communications, the client also embeds the related Task object, as specified in section 3.1.4.3, and submits the task communications for delivery, as specified in [MS-OXOMSG].

When the client accepts a task request, the client creates a local Task object and copies to it the relevant properties from the embedded Task object of the task request.

When the client receives a task request, it adds the sender to the **Cc recipients** list if the value of the PidLidTaskUpdates property is non-zero.

3.1.4.5 Receiving Updates

When a client receives a task response or a task update, it contains an embedded Task object, which is an update to a local Task object that client already has. The client uses the PidLidTaskGlobalId property of the embedded Task object to locate the local Task object (see [MS-OXCTABL] for details about using a **restriction** to find a Message object). If the client can locate the local Task object, it copies any relevant properties from the embedded Task object to the local Task object.

3.1.4.6   Task Communications

Before the client sends a task request, it computes the name of the new owner of the task by retrieving the primary recipients from the Task object. If there is only one primary recipient, its display name is the name of the new owner. If there are multiple primary recipients, the new owner name is derived by concatenating the display names of all the primary recipients, separated with semicolons (";"). The client sets the value of the PidLidTaskOwner property of the Task object with this new owner name. The client also sets the value of the PidLidTaskGlobalId property of the Task object to a new, unique GUID if it does not already have one.

When the client receives a task request, it appends an entry that represents the sender of the task request to the PidLidTaskAssigners property of the Task object and sets the value of the PidLidTaskOwner property of the Task object to the name of the task assignee. The client also adds the sender to the **Bcc recipients** of the Task object if the value of the PidLidTaskUpdates property of the Task object is non-zero.

Before the client sends a task response, it addresses the response to the last task assigner listed in the PidLidTaskAssigners property of the Task object.

Before the client sends a task rejection, it removes the last entry from the value of the PidLidTaskAssigners property of the Task object. The client sets the value of the PidLidTaskOwner property of the Task object to the name from this last entry.

3.1.4.7   Recipients in Task Objects

Clients do not submit task objects to servers for delivery to other users, even though they support recipients, as specified in [MS-OXCMSG]. Instead, clients embed task objects within task communications, as specified in section 3.1.4.6, for delivery to other users.

Yet, recipients are still meaningful for task objects. The client adds a user as a Cc recipient if that user wants to receive task updates. The client adds a user as a Bcc recipient if that user wants to receive an e-mail status report when the task is completed.

When the client changes a Task object, it sends a task update to all the Cc recipients if PidLidTaskUpdates is non-zero.

When the client marks a Task object as complete (by setting the value of the PidLidTaskStatus property), it sends an e-mail status report to all the Bcc recipients if PidLidTaskStatusOnComplete is non-zero.

Task requests can be assigned to one task assignee only. If a task request has more than one primary recipient, the Task object is shared, not assigned, and the client does not send task responses or task updates.

3.1.4.8   Generating Instances of Recurring Tasks

The client does not generate all instances of a recurring task at once. It begins by generating an initial instance only. In many cases, this instance will already exist when a recurrence pattern is added to it.

3.1.4.8.1   Deciding Whether to Generate a New Instance

The client considers generating a new instance of the recurring task when the prior instance: (a) is completed (the PidLidTaskStatus property is marked as Complete); (b) is deleted; or (c) is given a new recurring start date or due date.

While considering whether to generate a new instance of a recurring task, the client does not generate a new instance if the value of the PidLidTaskFRecurring property is 0x00 or if the value of the PidLidTaskDeadOccurrence property is 0x01.

The client also considers the criteria specified in the recurrence pattern. For details, see [MS-OXOCAL]. If the recurrence pattern specifies a valid end date and a positive count of occurrences, the client decrements the count of occurrences, saves the new recurrence pattern, and generates a new instance. If the occurrence count reaches 0, the client sets the value of the PidLidTaskDeadOccurrence property to 0x01.

3.1.4.8.2 New Instance Dates

Some recurrence patterns are "sliding," as specified in [MS-OXOCAL]. In such cases, the recurrence pattern does not specify the absolute date of each occurrence. Rather, the recurrence pattern specifies a date that is relative to the completion date of the prior instance. The client computes the date of the new instance accordingly.

Having determined from the recurrence pattern the appropriate date for a new instance, the client determines and sets the values for the start date and due date properties of the new instance. The new values of these properties are determined by combining the values of these properties from the prior instance with the newly calculated instance date, as follows: If the prior instance does not have a start date, the new instance does not have a start date, and the new due date is the newly calculated instance date. Otherwise, the new start date is the newly calculated instance date and the new instance and the new due date is the sum of the new start date and the difference between the old due date and the old start date. In other words, new due date = new start date + (old due date - old start date).

Finally, the client sets the reminder properties of the new instance, as specified in [MS-OXORMDR]. In particular, the client sets the PidLidReminderSet property of the new instance to 0x01 if the reminder time has not already passed and (a) the PidLidReminderSet property of the prior instance is 0x01, or (b) the PidLidTaskResetReminder property of the priori instance is 0x01. If the reminder time has already passed but either condition (a) or (b) applies, the client sets PidLidTaskResetReminder to 0x01 so that future instances can continue to follow the same logic.

3.1.4.8.3 Archive Instances

If a new instance is warranted, the client does not create a new Task object for the new instance. A new Task object would have distinct values for properties (PidTagSearchKey, PidTagEntryId, and others) that might affect later efforts to locate and identify the Task object. Instead, the client updates the properties of the existing Task object and uses it as the new **instance**. If preferred, the client first creates a new Task object to represent the now-completed task.

To create a Task object to represent the now-completed task, the client creates a new Task object, as usual. Then, the client copies any relevant recipients, attachments, and properties, as specified in [MS-OXCMSG], from the prior Task object to the new Task object, with these exceptions:

Property	Action
PidLidTaskOwnership	Set to 0, not assigned.
PidLidTaskAcceptanceState	Set to 0, not assigned.
PidLidTaskState	Set to 1, not assigned.
PidLidTaskMode	Set to 0, not assigned.

Property	Action
PidLidTaskOrdinal	Set to a unique ordinal.
PidTagReadReceiptRequested	Set to 0x00.
PidTagOriginatorDeliveryReportRequested	Set to 0x00.
PidLidTaskAssigner	Set to "", empty string.
PidTagSenderName	Delete.
PidTagSenderEmailAddress	Delete.
PidTagSenderAddressType	Delete.
PidTagSenderEntryId	Delete.
PidTagSenderSearchKey	Delete.
PidTagSentRepresentingName	Delete.
PidTagSentRepresentingEmailAddress	Delete.
PidTagSentRepresentingAddressType	Delete.
PidTagSentRepresentingEntryId	Delete.
PidTagSentRepresentingSearchKey	Delete.
PidLidTaskAssigners	Delete.
PidLidTaskFFixOffline	Set to 0x00.
PidLidTaskDeadOccurrence	Set to 0x01.
PidLidTaskStatus	Set to 2, complete.
PidLidTaskComplete	Set to 0x01.
PidLidPercentComplete	Set to 1.0, 100%.

3.1.4.9 Public Folders

Task objects in **public folders** are not assigned. That is, the client MUST NOT create task requests for task objects in public folders.

The client does not allow users to accept or reject task requests in public folders.

3.1.5 Message Processing Events and Sequencing Rules

None.

3.1.6 Timer Events

None.

3.1.7   Other Local Events

None.

3.2   Server Details

The server role for this protocol is as specified in [MS-OXCMSG].

4   Protocol Examples

The examples in the following sub-sections use both **named properties** and **tagged properties**. The **property ID** of a named property is provided by the server. Therefore, before setting or reading any properties of a Task object, the client asks the server to perform a mapping from **property names** (or **LIDs**) to property IDs by using the RopGetPropertyIdsFromNames operation (for details about this operation, see [MS-OXCPRPT] section 2.2.12).

The following table lists all of the named properties that are used in the examples.

Property	Property Set GUID	Property Name or LID
PidLidTaskComplete	{00062003-0000-0000-C000-000000000046}	0x0000811C
PidLidTaskStatus	{00062003-0000-0000-C000-000000000046}	0x00008101
PidLidPercentComplete	{00062003-0000-0000-C000-000000000046}	0x00008102
PidLidTaskActualEffort	{00062003-0000-0000-C000-000000000046}	0x00008110
PidLidTaskEstimatedEffort	{00062003-0000-0000-C000-000000000046}	0x00008111
PidLidTaskUpdates	{00062003-0000-0000-C000-000000000046}	0x0000811B
PidLidTaskStatusOnComplete	{00062003-0000-0000-C000-000000000046}	0x00008119
PidLidTaskFFixOffline	{00062003-0000-0000-C000-000000000046}	0x0000812C
PidLidTaskOwnership	{00062003-0000-0000-C000-000000000046}	0x00008129
PidLidTaskAcceptanceState	{00062003-0000-0000-C000-000000000046}	0x0000812A
PidLidTaskState	{00062003-0000-0000-C000-000000000046}	0x00008113
PidLidTaskOrdinal	{00062003-0000-0000-C000-000000000046}	0x00008123
PidLidTaskHistory	{00062003-0000-0000-C000-000000000046}	0x0000811A
PidLidTaskLastUpdate	{00062003-0000-0000-C000-000000000046}	0x00008115
PidLidTaskLastUser	{00062003-0000-0000-C000-000000000046}	0x00008122
PidLidTaskLastDelegate	{00062003-0000-0000-C000-000000000046}	0x00008125
PidLidTaskVersion	{00062003-0000-0000-C000-000000000046}	0x00008112
PidLidTaskOwner	{00062003-0000-0000-C000-000000000046}	0x0000811F
PidLidTaskFRecurring	{00062003-0000-0000-C000-000000000046}	0x00008126
PidLidTaskMode	{00062008-0000-0000-C000-000000000046}	0x00008518
PidLidTaskGlobalId	{00062008-0000-0000-C000-000000000046}	0x00008519
PidLidTaskDueDate	{00062003-0000-0000-C000-000000000046}	0x00008105
PidLidTaskStartDate	{00062003-0000-0000-C000-000000000046}	0x00008104
PidLidTaskDateCompleted	{00062003-0000-0000-C000-000000000046}	0x0000810F

Property	Property Set GUID	Property Name or LID
PidLidTaskAccepted	{00062003-0000-0000-C000-000000000046}	0x00008108
PidLidTaskResetReminder	{00062003-0000-0000-C000-000000000046}	0x00008107
PidLidTaskMultipleRecipients	{00062003-0000-0000-C000-000000000046}	0x00008120
PidLidTaskDeadOccurrence	{00062003-0000-0000-C000-000000000046}	0x00008109
PidLidTaskRole	{00062003-0000-0000-C000-000000000046}	0x00008127
PidLidTaskAssigners	{00062003-0000-0000-C000-000000000046}	0x00008117
PidLidTaskRecurrence	{00062003-0000-0000-C000-000000000046}	0x00008116
PidLidTaskAssigner	{00062003-0000-0000-C000-000000000046}	0x00008121
PidLidTaskFCreator	{00062003-0000-0000-C000-000000000046}	0x0000811E
PidLidCommonStart	{00062008-0000-0000-C000-000000000046}	0x00008516
PidLidCommonEnd	{00062008-0000-0000-C000-000000000046}	0x00008517

The server might respond with the following property IDs, which will be used in the examples (the actual property IDs are at the discretion of the server):

Property	Property ID
PidLidTaskComplete	0x8149
PidLidTaskStatus	0x8146
PidLidPercentComplete	0x8147
PidLidTaskActualEffort	0x814D
PidLidTaskEstimatedEffort	0x814E
PidLidTaskUpdates	0x82C3
PidLidTaskStatusOnComplete	0x82C4
PidLidTaskFFixOffline	0x8156
PidLidTaskOwnership	0x8154
PidLidTaskAcceptanceState	0x8151
PidLidTaskState	0x8148
PidLidTaskOrdinal	0x815D
PidLidTaskHistory	0x8150
PidLidTaskLastUpdate	0x8153
PidLidTaskLastUser	0x8152

Property	Property ID
PidLidTaskLastDelegate	0x82C5
PidLidTaskVersion	0x8158
PidLidTaskOwner	0x801B
PidLidTaskFRecurring	0x814B
PidLidTaskMode	0x8212
PidLidTaskGlobalId	0x8211
PidLidTaskDueDate	0x8145
PidLidTaskStartDate	0x8144
PidLidTaskDateCompleted	0x814A
PidLidTaskAccepted	0x82C2
PidLidTaskResetReminder	0x815C
PidLidTaskMultipleRecipients	0x814F
PidLidTaskDeadOccurrence	0x814C
PidLidTaskRole	0x8157
PidLidTaskAssigners	0x82C8
PidLidTaskRecurrence	0x815B
PidLidTaskAssigner	0x8159
PidLidTaskFCreator	0x82CA
PidLidCommonStart	0x81BD
PidLidCommonEnd	0x81BC

4.1 Sending a Task Request

Mary North assigns a task to her coworker, Paul West. The following is a description of what a client might do to accomplish Mary's intentions.

The client begins by obtaining property IDs from the server, as described in section 4.

To create the task request, the client uses the RopCreateMessage operation. The server returns a success code and a **handle** to a Message object. The client uses the RopSetProperties operation to transmit Mary's data to the server:

Property	Property ID	Type	Value
PidTagMessageClass	0x001A	0x001F (**PtypString**)	"IPM.TaskRequest"
PidTagIconIndex	0x1080	0x0003 (**PtypInteger32**)	0xFFFFFFFF

Property	Property ID	Type	Value
PidLidTaskMode	0x8212	0x0003 (**PtypInteger32**)	0x00000001

The client provides the actual Task object in an embedded Message object. The protocol creates an Attachment object into which it will embed the Task object by using the RopCreateAttachment operation, which returns a handle to the new Attachment object. The client then uses this handle with the RopSetProperties operation to set the PidTagAttachMethod property to afEmbeddedMessage (0x00000005):

Property	Property ID	Type	Value
PidTagAttachMethod	0x3705	0x0003 (**PtypInteger32**)	0x00000005 (afEmbeddedMessage)
PidTagRenderingPosition	0x370B	0x0003 (**PtypInteger32**)	0xFFFFFFFF
PidTagAttachmentHidden	0x7FFE	0x000B (**PtypBoolean**)	0x01

The client acquires the handle to the embedded Message object within the Attachment object by using the RopOpenEmbeddedMessage operation, which can be used as a Task object. The client sets the properties that it wants for this Task object or copies them from a local Task object by using the RopSetProperties operation:

Property	Property ID	Type	Value
PidTagMessageClass	0x001A	0x001F (**PtypString**)	"IPM.Task"
PidTagIconIndex	0x1080	0x0003 (**PtypInteger32**)	0x00000500
PidLidTaskComplete	0x8149	0x000B (**PtypBoolean**)	0x00
PidLidPercentComplete	0x8147	0x0005 (**PtypFloating64**)	0.0
PidLidTaskStatus	0x8146	0x0003 (**PtypInteger32**)	0 (Not started)
PidLidTaskActualEffort	0x814D	0x0003 (**PtypInteger32**)	0
PidLidTaskEstimatedEffort	0x814E	0x0003 (**PtypInteger32**)	0
PidLidTaskUpdates	0x82C3	0x000B (**PtypBoolean**)	0x01
PidLidTaskStatusOnComplete	0x82C4	0x000B (**PtypBoolean**)	0x01
PidLidTaskFFixOffline	0x8156	0x000B (**PtypBoolean**)	0x00

Property	Property ID	Type	Value
PidLidTaskOwnership	0x8154	0x0003 (**PtypInteger32**)	1 (assigner's copy)
PidLidTaskAcceptanceState	0x8151	0x0003 (**PtypInteger32**)	1 (unknown)
PidLidTaskState	0x8148	0x0003 (**PtypInteger32**)	3 (assigner's copy)
PidLidTaskOrdinal	0x815D	0x0003 (**PtypInteger32**)	–1000
PidLidTaskHistory	0x8150	0x0003 (**PtypInteger32**)	0x00000005 (sent with a task request)
PidLidTaskLastUpdate	0x8153	0x0040 (**PtypTime**)	2008/02/19 07:00
PidLidTaskLastUser	0x8152	0x001F (**PtypString**)	"Mary North"
PidLidTaskLastDelegate	0x82C5	0x001F (**PtypString**)	"Mary North"
PidLidTaskVersion	0x8158	0x0003 (**PtypInteger32**)	2
PidLidTaskOwner	0x801B	0x001F (**PtypString**)	"Paul West"
PidLidTaskFRecurring	0x814B	0x000B (**PtypBoolean**)	0x00
PidLidTaskMode	0x8212	0x0003 (**PtypInteger32**)	1 (embedded in a task request)
PidLidTaskGlobalId	0x8211	0x0102 (**PtypBinary**)	0E B0 1E 03 85 02 EF 4B 9A 14 50 83 B3 BB 4D E9
PidLidTaskMode	0x8212	0x0003 (**PtypInteger32**)	0x00000001

The client then sets other attachment properties, as specified in [MS-OXCMSG].

The client saves and closes the embedded Message object by using, in order, the following operations: RopSaveChangesMessage (embedded Task object handle), RopSaveChangesAttachment (attachment handle), RopRelease (embedded Task object handle), and RopRelease (attachment handle).

The client uses the **RopAddRecipients** operation to add Paul's recipient information to the task request. See [MS-OXOMSG] for details.

When Mary is ready to send her task request, the client uses the **RopSaveChanges** operation to commit the properties to the server, the RopSubmitMessage operation to send it, and then RopRelease to release the task request object.

4.2   Processing a Task Update

Russell King assigned a task to Scott Bishop. Scott updated some of the task properties, such as percent completed, and sent an update. Russell has now received a task update and needs to merge

Scott's changes into his own copy of the task. The following is a description of what a client might do to process the update.

The client begins by obtaining property IDs from the server as described in section 4.

The client obtains a handle to the task update by using the RopOpenMessage operation. The updated task information is part of the Task object that is embedded within the first attachment of the task update. To get the attachment, the client uses the handle to the task update with the RopOpenAttachment operation. The client gets a handle to the embedded Message object from this attachment by using the RopOpenEmbeddedMessage operation, which can then be used as the Task object. The client reads properties from the embedded Task object by using the RopGetPropertiesSpecific operation:

Property	Property ID	Type	Value obtained from server
PidLidTaskStatus	0x8146	0x0003 (**PtypInteger32**)	0 (Not started)
PidLidPercentComplete	0x8147	0x0005 (**PtypFloating64**)	0.0 (0%)
PidLidTaskDueDate	0x8145	0x0040 (**PtypTime**)	<not found>
PidLidTaskStartDate	0x8144	0x0040 (**PtypTime**)	<not found>
PidLidTaskActualEffort	0x814D	0x0003 (**PtypInteger32**)	0
PidLidTaskEstimatedEffort	0x814E	0x0003 (**PtypInteger32**)	0
PidLidTaskDateCompleted	0x814A	0x0040 (**PtypTime**)	<not found>
PidLidTaskAccepted	0x82C2	0x000B (**PtypBoolean**)	0x01
PidLidTaskResetReminder	0x815C	0x000B (**PtypBoolean**)	<not found>
PidLidTaskMultipleRecipients	0x814F	0x0003 (**PtypInteger32**)	0
PidLidTaskUpdates	0x82C3	0x000B (**PtypBoolean**)	0x01
PidLidTaskStatusOnComplete	0x82C4	0x000B (**PtypBoolean**)	0x01
PidLidTaskDeadOccurrence	0x814C	0x000B (**PtypBoolean**)	<not found>
PidLidTaskComplete	0x8149	0x000B (**PtypBoolean**)	0x00
PidLidTaskFFixOffline	0x8156	0x000B (**PtypBoolean**)	0x00
PidLidTaskOwnership	0x8154	0x0003	2 (task assignee's copy)

Property	Property ID	Type	Value obtained from server
		(**PtypInteger32**)	
PidLidTaskAcceptanceState	0x8151	0x0003 (**PtypInteger32**)	0 (Not assigned)
PidLidTaskHistory	0x8150	0x0003 (**PtypInteger32**)	3 (Updated)
PidLidTaskLastUpdate	0x8153	0x0040 (**PtypTime**)	2008/02/19
PidLidTaskLastUser	0x8152	0x001F (**PtypString**)	"Scott Bishop"
PidLidTaskLastDelegate	0x82C5	0x001F (**PtypString**)	"Scott Bishop"
PidLidTaskRole	0x8157	0x001F (**PtypString**)	""
PidLidTaskVersion	0x8158	0x0003 (**PtypInteger32**)	4
PidLidTaskState	0x8148	0x0003 (**PtypInteger32**)	2 (task assignee's copy)
PidLidTaskAssigners	0x82C8	0x0102 (**PtypBinary**)	<binary data>
PidLidTaskRecurrence	0x815B	0x0102 (**PtypBinary**)	<not found>
PidLidTaskAssigner	0x8159	0x001F (**PtypString**)	"Russell King"
PidLidTaskOwner	0x801B	0x001F (**PtypString**)	"Scott Bishop"
PidLidTaskFCreator	0x82CA	0x000B (**PtypBoolean**)	0x00
PidLidTaskOrdinal	0x815D	0x0003 (**PtypInteger32**)	–1000
PidLidTaskFRecurring	0x814B	0x000B (**PtypBoolean**)	0x00
PidLidCommonStart	0x81BD	0x0040 (**PtypTime**)	<not found>
PidLidCommonEnd	0x81BC	0x0040 (**PtypTime**)	<not found>
PidLidTaskGlobalId	0x8211	0x0102 (**PtypBinary**)	0E B0 1E 03 85 02 EF 4B 9A 14 50 83 B3 BB 4D E9

The client will use the value of the PidLidTaskGlobalId property to locate the Task object locally and will then use the values of the other properties to copy to the local Task object.

The client uses a handle to the Tasks special folder and the RopGetContentsTable operation to get a handle to the **contents table** of the folder. Using this handle, the client uses the RopSetColumns operation:

Property	Property ID	Type
PidTagFolderId	0x6748	0x0014 (**PtypInteger64**)
PidTagMid	0x674A	0x0014 (**PtypInteger64**)

With the proper column set, the client can now search for the local Task object whose PidLidTaskGlobalId property matches the one found in the embedded Task object. The client performs the search with the RopFindRow operation:

Condition Type	Relational Operator	Property ID	Property Data
0x04 (RES_PROPERTY)	0x04 (RELOP_EQ)	0x8211 (PidLidTaskGlobalId)	`0E B0 1E 03 85 02 EF 4B 9A 14 50 83 B3 BB 4D E9`

Having completed the search, the client releases the handle to the contents table by using RopRelease.

If the search succeeded, the client will have located the PidTagFolderId and PidTagMid properties for the local Task object. The client uses these values to open a handle to the local Task object by using RopOpenMessage. The client will now use the RopSetProperties operation to update the properties of the local Task object, copying the properties from the embedded Task object, as appropriate:

Property	Property ID	Type	Value
PidLidTaskStatus	0x8146	0x0003 (**PtypInteger32**)	`0` (Not started)
PidLidPercentComplete	0x8147	0x0005 (**PtypFloating64**)	`0.0` (0%)
PidLidTaskActualEffort	0x814D	0x0003 (**PtypInteger32**)	`0`
PidLidTaskEstimatedEffort	0x814E	0x0003 (**PtypInteger32**)	`0`
PidLidTaskAccepted	0x82C2	0x000B (**PtypBoolean**)	`0x01`
PidLidTaskMultipleRecipients	0x814F	0x0003 (**PtypInteger32**)	`0`
PidLidTaskUpdates	0x82C3	0x000B (**PtypBoolean**)	`0x01`
PidLidTaskStatusOnComplete	0x82C4	0x000B (**PtypBoolean**)	`0x01`
PidLidTaskComplete	0x8149	0x000B (**PtypBoolean**)	`0x00`
PidLidTaskFFixOffline	0x8156	0x000B (**PtypBoolean**)	`0x00`

Property	Property ID	Type	Value
PidLidTaskOwnership	0x8154	0x0003 (**PtypInteger32**)	1 (task assigner's copy)
PidLidTaskAcceptanceState	0x8151	0x0003 (**PtypInteger32**)	2 (Accepted)
PidLidTaskHistory	0x8150	0x0003 (**PtypInteger32**)	1 (Accepted)
PidLidTaskLastUpdate	0x8153	0x0040 (**PtypTime**)	2008/02/19
PidLidTaskLastUser	0x8152	0x001F (**PtypString**)	"Scott Bishop"
PidLidTaskLastDelegate	0x82C5	0x001F (**PtypString**)	"Scott Bishop"
PidLidTaskRole	0x8157	0x001F (**PtypString**)	""
PidLidTaskVersion	0x8158	0x0003 (**PtypInteger32**)	4
PidLidTaskState	0x8148	0x0003 (**PtypInteger32**)	3 (task assigner's copy of an accepted Task object)
PidLidTaskAssigner	0x8159	0x001F (**PtypString**)	""
PidLidTaskOwner	0x801B	0x001F (**PtypString**)	"Scott Bishop"
PidLidTaskFCreator	0x82CA	0x000B (**PtypBoolean**)	0x01
PidLidTaskOrdinal	0x815D	0x0003 (**PtypInteger32**)	–1000
PidLidTaskFRecurring	0x814B	0x000B (**PtypBoolean**)	0x00
PidLidTaskGlobalId	0x8211	0x0102 (**PtypBinary**)	0E B0 1E 03 85 02 EF 4B 9A 14 50 83 B3 BB 4D E9

The client saves and closes the local Task object, embedded Task object, and **attachment** by using, in order, RopSaveChangesMessage (local Task object handle), RopRelease (embedded Task object handle), RopRelease (attachment handle), and RopRelease (local Task object handle).

5   Security

5.1   Security Considerations for Implementers

There are no special security considerations specific to the Task-Related Objects Protocol. General security considerations pertaining to the underlying transport apply, as specified in [MS-OXCMSG] and [MS-OXCPRPT].

5.2   Index of Security Parameters

None.

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 2.2: Outlook 2003, Outlook 2007, and Outlook 2010 set the following properties regardless of user input; their values have no meaning in the context of this protocol.PidLidAgingDontAgeMe, PidLidCurrentVersion, PidLidCurrentVersionName, PidLidPrivate, PidLidSideEffects, PidLidValidFlagStringProof, PidTagAlternateRecipientAllowed, PidTagClientSubmitTime, PidTagDeleteAfterSubmit, PidTagImportance, PidTagMessageDeliveryTime, PidTagMessageLocaleId, PidTagNormalizedSubject, PidTagOriginatorDeliveryReportRequested, PidTagPriority, PidTagReadReceiptRequested, PidTagSensitivity.

<2> Section 2.2: Outlook 2003, Outlook 2007, and Outlook 2010 set the following properties on task objects regardless of user input; their values have no meaning in the remaining contexts of this protocol.PidLidTaskNoCompute, PidLidTaskRole, PidLidTaskCustomFlags, PidLidTeamTask

<3> Section 2.2: Outlook 2007 sets the following properties regardless of user input; their values have meaning in the context of this protocol only when applied to task objects.PidLidPercentComplete, PidLidTaskActualEffort, PidLidTaskComplete, PidLidTaskAssigner, PidLidTaskAcceptanceState, PidLidTaskEstimatedEffort, PidLidTaskFFixOffline, PidLidTaskFRecurring, PidLidTaskOrdinal, PidLidTaskOwnership, PidLidTaskState, PidLidTaskStatus.

<4> Section 2.2.2.2.16: Outlook 2003, Outlook 2007, and Outlook 2010 sometimes delete the stack of assigners incorrectly, leaving only the most recent assigner.

<5> Section 2.2.2.2.23: Outlook 2003, Outlook 2007, and Outlook 2010 sometimes set this property to indicate that the Task object is a "Team Task." However, the distinction is no longer meaningful.

<6> Section 3.1.4.3: Outlook 2003, Outlook 2007, and Outlook 2010 set the rendering position and hidden flag in a separate RopSetProperties request, after opening the embedded message.

7 Change Tracking

This section identifies changes made to [MS-OXOTASK] protocol documentation between February 2010 and May 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.

- 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
1.3 Overview	Updated the section title.	N	Content updated for template compliance.

Release: Saturday, May 1, 2010

8 Index