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



| Revision Summary | | | |
|---|---|---|---|
| Author | Date | Version | Comments |
| Microsoft Corporation | April 4, 2008 | 0.1 | Initial Availability. |
| Microsoft Corporation | April 25, 2008 | 0.2 | Revised and updated property names and other technical content. |
| Microsoft Corporation | June 27, 2008 | 1.0 | Initial Release. |
| Microsoft Corporation | August 6, 2008 | 1.01 | Revised and edited technical content. |

# Table of Contents

# 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]

**EntryID**
**Attachment object**
**Bcc recipient**
**Cc recipient**
**contents table**
**Folder object**
**GUID**
**handle**
**Message object**
**primary recipient**
**property (1)**
**public folder**
**recipient**
**recurrence pattern**
**reminder**
**remote operation (ROP)**
**special folder**
**Task object**
**task request**
**Unicode**
**Coordinated Universal Time (UTC)**

The following terms are specific to this document:

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

**recurring task:** A series of tasks that are described by a **recurrence pattern**.

**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 communications:** Collectively, **task requests**, **task acceptances**, **task rejections**, and **task updates**.

**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 responses:** Collectively, **task acceptances** and **task rejections**.

**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

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

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

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

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

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

[MS-OXOMSG] Microsoft Corporation, "E-mail Object Protocol Specification", June 2008.

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

[MS-OXPROPS] Microsoft Corporation, "Office Exchange Protocols Master Property List Specification", 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 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 Task objects and task communications by using the Message and Attachment Object protocol, as specified in [MS-OXCMSG]. How a server operates on Task objects and task communications is implementation-dependent, but the results of any such operations have to 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 adhere to all property constraints specified in [MS-OXPROPS] and all property constraints specified in [MS-OXCMSG]. A Task object and a task communication MAY <1> <2> <3> also contain other properties, which are specified in [MS-OXPROPS], but these properties have no impact on the Task-Related Objects protocol.

### 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 this **Task object**. The value MUST be 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 MUST be 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**. |
| 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 MUST be set to 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 0x00. This value is set to 0x01 when the task's reminder is dismissed, and set to 0x00 otherwise. If left unset, a default of 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 0x01. At the appropriate time, the client presents the user with a reminder. When the user dismisses the reminder, the client sets **PidLidReminderSet** to 0x00 (and sets **PidLidTaskResetReminder** to 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 0x00. The client uses the 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 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-OXCRMDR], if either **PidLidReminderSet** or **PidLidTaskResetReminder** is 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 0x00 for a new Task object and to 0x01 when a Task object is either accepted or rejected. If left unset, a value of 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 0x00 for a new Task object and to 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 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. MAY 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.

#### 2.2.2.2.14 PidLidTaskState

Type: **PtypInteger32**

Indicates the current assignment state of the **Task object**; MUST be 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**. 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** | *EID* | Task assigner's Address Book EntryID. |
| Variable | **PtypString8** | *szDisplayName* | Task assigner's **display name**, using non-Unicode characters. |
| Variable | **PtypString**, as Unicode | *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.

#### 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 MUST be 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.

#### 2.2.2.2.20 PidLidTaskComplete

Type: **PtypBoolean**

Indicates that the task has been completed.

#### 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 to 0x01 when the user creates the task and to 0x00 when the task was assigned by another user. If left unset, a value of 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 MAY 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 section 2.2.2.2.16), this property MUST be 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** MAY 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 user who most recently assigned the task, or the user to whom it was most recently assigned.

Before sending a **task request**, the client sets this **property** to the name of the **task assigner**. Before sending a **task response**, the client sets this property to the name of the **task assignee**.

#### 2.2.2.2.28 PidLidTaskFRecurring

Type: **PtypBoolean**

Indicates whether the task includes a recurrence pattern. If left unset, a default value of 0x00 is assumed. If set to 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**. MUST be one of the following values:

| 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. MUST be one of the following values:

| 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 the accuracy of **PidLidTaskOwner**.

| Value | Meaning |
|---|---|
| 0x00 or unset | The value for **PidLidTaskOwner** is correct. |
| 0x01 | The client cannot determine an accurate value for **PidLidTaskOwner**. |

When setting a value of 0x01, the client MAY also set the **PidLidTaskOwner** property to a generic owner name, such as "Unknown".

A client that discovers a value of 0x01 in this property and that can determine an accurate owner name updates **PidLidTaskOwner** and sets the value of this property to 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 is 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 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 MUST be 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 this **task communication**. The value MUST be 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

The client role is to create and manipulate **Task objects**, and otherwise the client operates in its roles as specified in [MS-OXCMSG].

### 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** 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 section 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 section 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 [MX-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 recipient**s 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. |
| **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 does 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-OXCPRPT].

# 4 Protocol Examples

All the examples that follow use property identifiers that are provided by the server. The client asks the server to perform a mapping from property names to property identifiers by using the **RopGetPropertyIdsFromNames** operation (for details about this operation, see [MS-OXCPRPT] section 2.2.12).

| Property | Property Set GUID | Name or ID |
|---|---|---|
| **PidLidTaskComplete** | {00062003-0000-0000-C000-00000000046} | 0x0000811C |
| **PidLidTaskStatus** | {00062003-0000-0000-C000-00000000046} | 0x00008101 |
| **PidLidPercentComplete** | {00062003-0000-0000-C000-00000000046} | 0x00008102 |
| **PidLidTaskActualEffort** | {00062003-0000-0000-C000-00000000046} | 0x00008110 |

| | | |
|---|---|---|
| **PidLidTaskEstimatedEffort** | {00062003-0000-0000-C000-00000000046} | 0x00008111 |
| **PidLidTaskUpdates** | {00062003-0000-0000-C000-00000000046} | 0x0000811B |
| **PidLidTaskStatusOnComplete** | {00062003-0000-0000-C000-00000000046} | 0x00008119 |
| **PidLidTaskFFixOffline** | {00062003-0000-0000-C000-00000000046} | 0x0000812C |
| **PidLidTaskOwnership** | {00062003-0000-0000-C000-00000000046} | 0x00008129 |
| **PidLidTaskAcceptanceState** | {00062003-0000-0000-C000-00000000046} | 0x0000812A |
| **PidLidTaskState** | {00062003-0000-0000-C000-00000000046} | 0x00008148 |
| **PidLidTaskOrdinal** | {00062003-0000-0000-C000-00000000046} | 0x00008123 |
| **PidLidTaskHistory** | {00062003-0000-0000-C000-00000000046} | 0x0000811A |
| **PidLidTaskLastUpdate** | {00062003-0000-0000-C000-00000000046} | 0x00008115 |
| **PidLidTaskLastUser** | {00062003-0000-0000-C000-00000000046} | 0x00008122 |
| **PidLidTaskLastDelegate** | {00062003-0000-0000-C000-00000000046} | 0x00008125 |
| **PidLidTaskVersion** | {00062003-0000-0000-C000-00000000046} | 0x00008112 |
| **PidLidTaskOwner** | {00062003-0000-0000-C000-00000000046} | 0x0000811F |
| **PidLidTaskFRecurring** | {00062003-0000-0000-C000-00000000046} | 0x00008126 |
| **PidLidTaskMode** | {00062008-0000-0000-C000-00000000046} | 0x00008518 |
| **PidLidTaskGlobalId** | {00062008-0000-0000-C000-00000000046} | 0x00008519 |

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

| Property | Property Identifier |
|---|---|
| **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 |
| **PidLidTaskLastDelegate** | 0x82C5 |
| **PidLidTaskVersion** | 0x8158 |
| **PidLidTaskOwner** | 0x801B |
| **PidLidTaskFRecurring** | 0x814B |
| **PidLidTaskMode** | 0x8212 |
| **PidLidTaskGlobalId** | 0x8211 |

## 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 identifiers 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 Identifier | Type | Value |
|---|---|---|---|
| **PidTagMessageClass** | 0x001A | 0x001F (**PTypString**) | "IPM.TaskRequest" |
| **PidTagIconIndex** | 0x1080 | 0x0003 (**PTypInteger32**) | 0xFFFFFFFF |

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 Identifier | Type | Value |
|---|---|---|---|
| | | | |

| | | | |
|---|---|---|---|
| **PidTagAttachMethod** | 0x3705 | 0x0003 (**PTypInteger32**) | 0x00000005 (afEmbeddedMessage) |
| **PidTagRenderingPosition** | 0x370B | 0x0040 (**PTypTime**) | 4501/01/01 |
| **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 Identifier | 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 |
| **PidLidTaskOwnership** | 0x8154 | 0x0003 (**PTypInteger32**) | 0 (Not assigned) |
| **PidLidTaskAcceptanceState** | 0x8151 | 0x0003 (**PTypInteger32**) | 0 (Not assigned) |
| **PidLidTaskState** | 0x8148 | 0x0003 (**PTypInteger32**) | 1 (Not assigned) |
| **PidLidTaskOrdinal** | 0x815D | 0x0003 (**PTypInteger32**) | -1000 |

| | | | |
|---|---|---|---|
| **PidLidTaskHistory** | 0x8150 | 0x0003 (**PTypInteger32**) | `0x00000005` (sent with a task request) |
| **PidLidTaskLastUpdate** | 0x8153 | 0x0040 (**PTypTime**) | `2008/02/19` |
| **PidLidTaskLastUser** | 0x8152 | 0x001F (**PTypString**) | "`Mary North`" |
| **PidLidTaskLastDelegate** | 0x82C5 | 0x001F (**PTypString**) | "`Mary North` " |
| **PidLidTaskVersion** | 0x8158 | 0x0003 (**PTypInteger32**) | `1` |
| **PidLidTaskOwner** | 0x801B | 0x001F (**PTypString**) | "`Mary North`" |
| **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` |

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 identifiers 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 Identifier | 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 (**PTypInteger32**) | `2` (**task assignee's** copy) |
| **PidLidTaskAcceptanceState** | 0x8151 | 0x0003 (**PTypInteger32**) | `0` (Not assigned) |
| **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**) | `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 Identifier | 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 Identifier | 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 Identifier | 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 |
| **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: Office/Exchange Behavior

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

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

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

1 Outlook 2003 SP3 and Outlook 2007 SP1 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 Outlook 2003 SP3 and Outlook 2007 SP1 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 Outlook 2007 SP1 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 Outlook 2003 SP3 and Outlook 2007 SP1 sometimes delete the stack of assigners incorrectly, leaving only the most recent assigner.

5 Outlook 2003 SP3 and Outlook 2007 SP1 sometimes set this property to indicate that the **Task object** is a "Team Task." However, the distinction is no longer meaningful.

6 Outlook 2003 SP3 and Outlook 2007 SP1 set the rendering position and hidden flag in a separate **ROPSetProperties** request, after opening the embedded message.

# Index