[MS-CMP]:
MSDTC Connection Manager:
OleTx Multiplexing Protocol Specification

This protocol documentation is intended for use in conjunction with publicly available standard specifications, network programming art, and Microsoft Windows distributed systems concepts, and assumes that the reader either is familiar with the aforementioned material or has immediate access to it.

A protocol specification does not require the use of Microsoft programming tools or programming environments in order for you to develop an implementation. If you have access to Microsoft programming tools and environments you are free to take advantage of them.

Revision Summary

Date	Revision History	Revision Class	Comments
04/03/2007	1.0		MCPP Milestone Longhorn Initial Availability
07/03/2007	2.0	Major	MLonghorn+90
07/20/2007	3.0	Major	Updated and revised the technical content.
08/10/2007	3.0.1	Editorial	Revised and edited the technical content.

Date	Revision History	Revision Class	Comments
09/28/2007	4.0	Major	Made a change to the IDL.
10/23/2007	5.0	Major	Updated and revised the technical content.
11/30/2007	5.0.1	Editorial	Revised and edited the technical content.
01/25/2008	5.0.2	Editorial	Revised and edited the technical content.

Table of Contents

1 Introduction

This document specifies the MSDTC Connection Manager: OleTx Multiplexing Protocol.

1.1 Glossary

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

Acceptor
Boxcar
Connection
Initiator
Little-Endian
Protocol Type
Session
Unicode

The following terms are specific to this document:

Name Object: An object that contains contact information specific to the MSDTC Connection Manager: OleTx Transports Protocol, as specified in [MS-CMPO].

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-CMPO] Microsoft Corporation, "MSDTC Connection Manager: OleTx Transports Protocol Specification", July 2007.

[MS-ERREF] Microsoft Corporation, "Windows Error Codes", January 2007.

[MS-GLOS] Microsoft Corporation, "Windows Protocols Master Glossary", March 2007.

[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

[MS-DTCO] Microsoft Corporation, "MSDTC Connection Manager: OleTx Transaction Protocol Specification", July 2007.

1.3 Protocol Overview (Synopsis)

The MSDTC Connection Manager: OleTx Multiplexing Protocol allows partners to multiplex any number of two-way **connections** over the MSDTC Connection Manager: OleTx Transports Protocol

session between them. To do this, the protocol defines a small number of messages to manage connections and uses the MSDTC Connection Manager: OleTx Transports Protocol resource requests to allocate connection-related resources. To facilitate higher-level protocols, the MSDTC Connection Manager: OleTx Multiplexing Protocol defines a single user message and allows associating a **protocol type** with a connection.

To illustrate these concepts, the following figure depicts typical MSDTC Connection Manager: OleTx Multiplexing Protocol messages to initiate, use, and terminate two connections between partners labeled A and B.

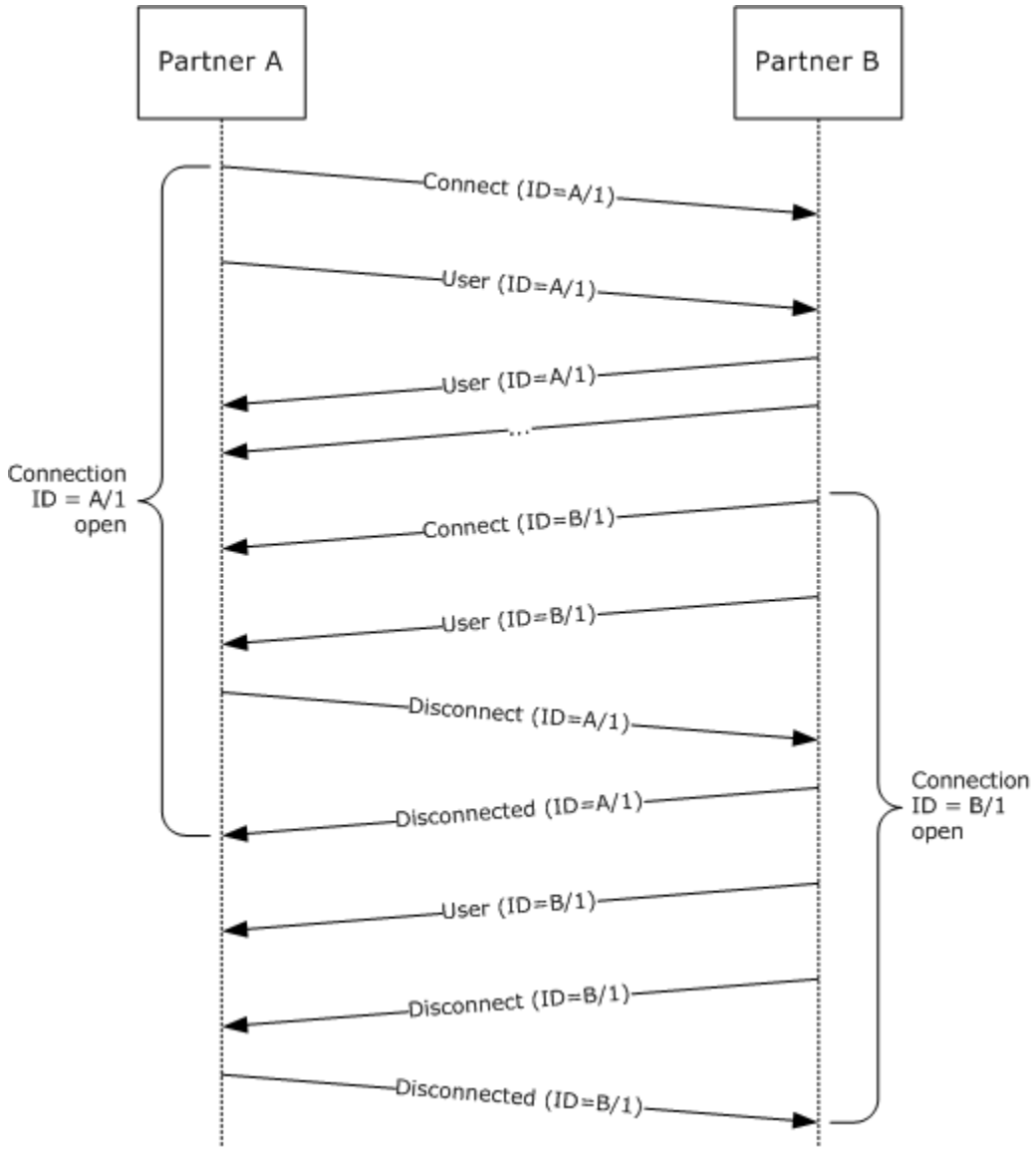


Figure 1: Messages used to manage two connections between partners

As the first message of the preceding figure depicts, to initiate a connection, either partner sends a Connect message (MTAG_CONNECTION_REQ) to the other partner over their session.

A Connect message includes an identifier for the new connection (abbreviated ID in the figure). To simplify connection management, connections are identified by two pieces of information: the partner that initiated the connection, and an identifier assigned by that partner. This scheme allows each partner to assign identifiers without the risk of collision with the other partner. In effect, each partner maintains two tables of connections: those initiated by itself (so-called "outgoing" connections) and those initiated by the other partner (so-called "incoming" connections). Despite the names "outgoing" and "incoming," either partner has the option to send messages to the other by using any open connection. To correlate a message with its connection, the message includes a flag (fIsMaster) indicating which table the connection belongs to, in addition to the initiator-assigned identifier (dwConnectionId) for the connection.

Though not depicted in the figure, a Connect message also includes a type to identify the higher-level protocol for the connection's messages. Specifically, this protocol type typically implies which types of User messages are expected over the connection.

As depicted in the preceding figure, a Connect message is assumed to succeed. If the receiving partner does not want to accept the connection, it sends a not-acknowledged (NACK) message (MSG_CONNECTION_REQ_DENIED).

After a connection is open, either partner has the option to send any number of User messages (MTAG_USER_MESSAGE) to the other partner by using that connection. User messages include their connection, a message subtype (typically defined by a higher-level protocol), and the message payload. As the receiving partner never sends positive acknowledgment to a Connect message, the sending partner is free to send User messages to the connection along with the Connect message. If the receiving partner does not accept the connection then it will ignore these extraneous User messages.

A partner receives messages in the order in which they were sent over the connection.

To close a connection, the partner that initiated the connection sends a Disconnect message (MTAG_DISCONNECT) to the other partner; either partner has the option to initiate a connection, but only the partner that initiated a connection is allowed to close it. Unlike the Connect message, the Disconnect message is assumed to fail. As the preceding figure depicts, if the receiving partner has the option to close the connection, it does so and sends a Disconnect acknowledgment (ACK) message (MTAG_DISCONNECTED). Finally, on receipt of the Disconnected message, the initiating partner closes the connection on its side. This asymmetric design avoids a race condition and allows the receiving partner to send any outstanding messages to the initiating partner before acknowledging the Disconnected message.

For efficiency, the MSDTC Connection Manager: OleTx Multiplexing Protocol batches messages by using Boxcar objects that contain one or more messages for one or more connections. A **Boxcar** includes the number of messages it encloses, their total size, and the messages themselves. Typically, the fact that messages are enclosed in a Boxcar is transparent to connection management and User messages in the MSDTC Connection Manager: OleTx Multiplexing Protocol. One exception occurs when a partner receives an unrecognized message type and discards the rest of the messages in the Boxcar.

1.4 Relationship to Other Protocols

This protocol is explicitly layered upon the MSDTC Connection Manager: OleTx Transports Protocol, and its design is greatly influenced by that protocol. It relies on the MSDTC Connection Manager: OleTx Transports Protocol to provide sessions and peer-to-peer message exchange. This protocol, in turn, provides message batching and connection multiplexing services to a protocol layered above it. For example, the MSDTC Connection Manager: OleTx Transaction Protocol protocol is a set of connections with different protocol types layered above the MSDTC Connection Manager: OleTx Multiplexing Protocol, and it is used for coordinating distributed atomic transactions.

1.5 Prerequisites/Preconditions

It is assumed that two MSDTC Connection Manager: OleTx Transports Protocol partners that want to communicate with each other are already aware of this fact through some other mechanism; there is no handshake or initialization to communicate this fact between MSDTC Connection Manager: OleTx Multiplexing Protocol instances.

1.6 Applicability Statement

This protocol is suitable for use as a connection multiplexing protocol over the MSDTC Connection Manager: OleTx Transports Protocol, and it is applicable in all of the same situations.

1.7 Versioning and Capability Negotiation

There are no optional capabilities exposed by the MSDTC Connection Manager: OleTx Multiplexing Protocol, and there are no extensibility points within the MSDTC Connection Manager: OleTx Multiplexing Protocol. There are therefore no version negotiation capabilities in this protocol.

1.8 Vendor-Extensible Fields

This protocol uses HRESULTs as defined in [MS-ERREF]. Vendors are free to choose their own values, as long as the C bit (0x20000000) is set, indicating that it is a customer code.

1.9 Standards Assignments

The MSDTC Connection Manager: OleTx Multiplexing Protocol does not use any standard assignments.

2 Messages

This section specifies how the MSDTC Connection Manager: OleTx Multiplexing Protocol messages are encapsulated on the wire and common data types.

2.1 Transport

Messages in the MSDTC Connection Manager: OleTx Multiplexing Protocol MUST be transported over an MSDTC Connection Manager: OleTx Transports Protocol session; therefore, each MSDTC Connection Manager: OleTx Multiplexing Protocol instance MUST have an underlying MSDTC Connection Manager: OleTx Transports Protocol instance. The initialization of the MSDTC Connection Manager: OleTx Transports Protocol instance occurs during the initialization of the MSDTC Connection Manager: OleTx Multiplexing Protocol instance, and is specified in section 3.

2.1.1 Transmitting Messages and Boxcars

Every message in MSDTC Connection Manager: OleTx Multiplexing Protocol is a subtype of the MESSAGE_PACKET (section 2.2.2) structure. When any event causes an MSDTC Connection Manager: OleTx Multiplexing Protocol implementation to send a message, an MSDTC Connection Manager: OleTx Multiplexing Protocol implementation MUST place this message in a boxcar. Boxcars are represented conceptually as Boxcar objects in the abstract data model; adding a message to a boxcar is represented conceptually as adding a message to the end of the Message List in a Boxcar object. For more information about Boxcar objects in the abstract data model, see section 3.1.1.2. For more information about processing boxcars, see section 3.1.5.

2.1.1.1 Boxcar Format

A boxcar is formatted as an array of bytes that begins with a BOX_CAR_HEADER (section 2.2.1) structure, and continues with one or more MESSAGE_PACKET structures. Each MESSAGE_PACKET structure in a boxcar MUST be aligned on an 8-byte boundary. Because the size of each MESSAGE_PACKET structure is a multiple of 4 bytes (as defined in section 2.2.2), padding bytes (non-deterministic values) MUST be added as necessary between the structures in order to have each structure aligned on a 8-byte boundary. The **dwcMessages** field of the BOX_CAR_HEADER structure MUST match the number of messages in the boxcar, and the **dwcbTotal** field of the BOX_CAR_HEADER structure MUST be equal to the total number of bytes in the boxcar.

2.1.1.2 Boxcar Size Limitations

A boxcar MUST contain at least one message, and MUST NOT contain more than 4,095 messages. Furthermore, the total size of a boxcar MUST be at least 32 bytes, and MUST NOT exceed 81,920 bytes. Unless otherwise specified, an MSDTC Connection Manager: OleTx Multiplexing Protocol implementation MAY add a message to a boxcar so long as doing so does not cause the boxcar to exceed any of these size restrictions.<1>

2.1.1.3 Transmitting Boxcars

When an MSDTC Connection Manager: OleTx Multiplexing Protocol implementation wants to transmit a boxcar on a session, the MSDTC Connection Manager: OleTx Multiplexing Protocol provides its underlying MSDTC Connection Manager: OleTx Transports Protocol implementation with the session to transmit the boxcar on, the count of messages in the boxcar, and the byte array that makes up the boxcar itself, as specified in MS-CMPO section 3.2.6.5. Also, as specified in that section, an MSDTC Connection Manager: OleTx Multiplexing Protocol implementation MUST NOT transmit more than one boxcar at a time.

For more information about transmitting messages in boxcars, see section 3.1.7.1. For more information about interpreting boxcars after they have been received, see section 3.1.5.

2.1.2 Security

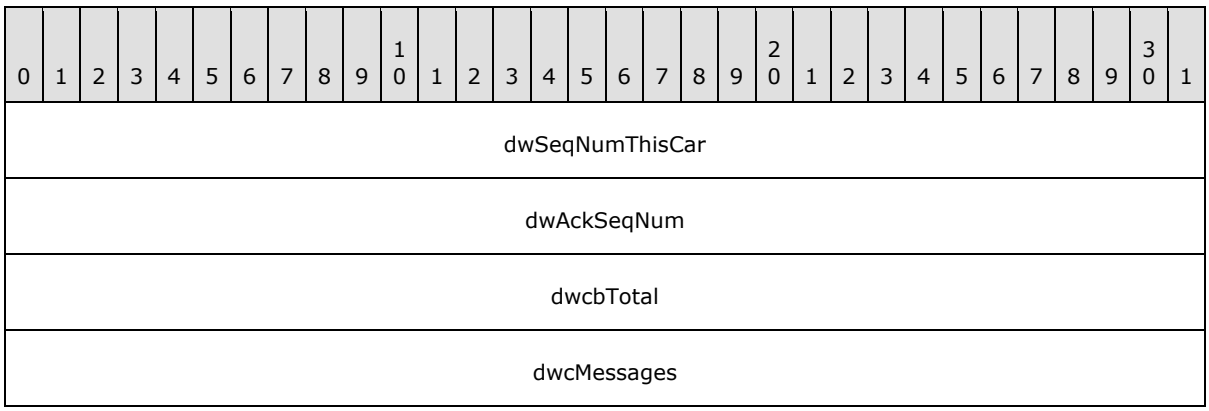
The MSDTC Connection Manager: OleTx Multiplexing Protocol does not introduce any additional authentication or authorization capabilities beyond those provided by the MSDTC Connection Manager: OleTx Transports Protocol. The authentication and authorization capabilities exposed by the MSDTC Connection Manager: OleTx Transports Protocol MUST be directly exposed to any higher-layer protocols for them to take advantage of.

2.2 Message Syntax

All integer fields in the following structures are in **little-endian** byte order, and all structures MUST be aligned with an 8-byte alignment. Any padding bytes that are required to align the MESSAGE_PACKET structures within the boxcar SHOULD be set to 0 by the sender and MUST be ignored by the receiver.<2>

2.2.1 BOX_CAR_HEADER

The BOX_CAR_HEADER structure MUST be the first structure in each boxcar transmitted via the underlying MSDTC Connection Manager: OleTx Transports Protocol session.



dwSeqNumThisCar (4 bytes): This field is not used, it MUST be set to 0, and it MUST be ignored on receipt.

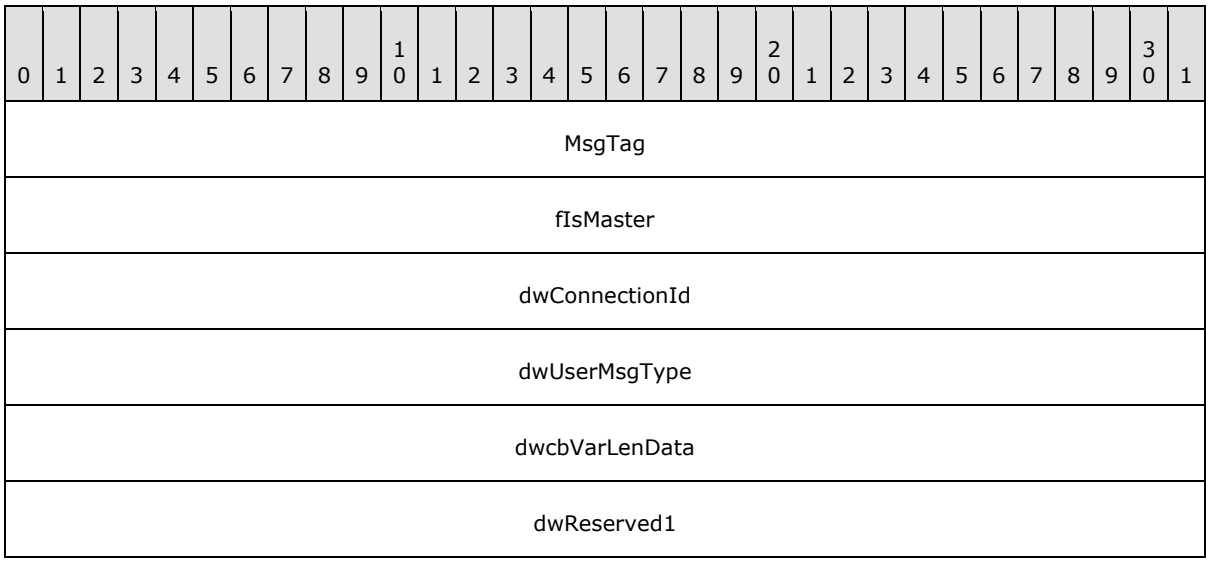
dwAckSeqNum (4 bytes): This field is not used, it MUST be set to 0, and it MUST be ignored on receipt.

dwcbTotal (4 bytes): The total number of bytes in the message packet, including the header. This value MUST be equal to the size of the boxcar. It MUST be greater than or equal to 32, and it MUST be less than or equal to 81,920.

dwcMessages (4 bytes): The number of Message_Packet structures that follow the end of this structure in the boxcar. This number MUST be greater than or equal to 1, and MUST BE less than or equal to 4,095.

2.2.2 MESSAGE_PACKET

Each message sent using the MSDTC Connection Manager: OleTx Multiplexing Protocol MUST be an extension of the MESSAGE_PACKET structure. This structure forms the basis for all of these messages. All integer fields of this structure are in little-endian byte order.



MsgTag (4 bytes): A 4-byte integer value that describes the message type and its interpretation. This value MUST be one of the following values.

Value	Meaning
MTAG_DISCONNECT 0x00000001	Indicates a request to disconnect the specified connection.
MTAG_DISCONNECTED 0x00000002	Indicates that the specified connection has been disconnected.
MTAG_CONNECTION_REQ_DENIED 0x00000003	Indicates that the connection request for the specified connection has been denied.
MTAG_PING 0x00000004	Indicates that the session should continue to be maintained.
MTAG_CONNECTION_REQ 0x00000005	Indicates that a new connection is being requested.
MTAG_USER_MESSAGE 0x00000FFF	Indicates that a user message should be delivered on the specified connection.

If the value is not one of the preceding values, then the remainder of the boxcar MUST be discarded. The details of each message type are given in the following sections.

fIsMaster (4 bytes): A 4-byte value indicating the direction of the message in the conversation. This value MUST be one of the following values.

Value	Meaning
0x00000000	Message is sent by the party that accepted the connection.
0x00000001	Message is sent by the party that initiated the connection.

dwConnectionId (4 bytes): A 4-byte integer value that contains the unique identifier for the associated connection. The value of the identifier depends on the value of the **MsgTag** field, as follows.

MsgTag field value	dwConnectionId field
0x00000001 MTAG_DISCONNECT	MUST contain the ID of the connection being disconnected.
0x00000002 MTAG_DISCONNECTED	MUST contain the ID of the connection that was just disconnected.
0x00000003 MTAG_CONNECTION_REQ_DENIED	MUST contain the ID of the connection that was rejected.
0x00000004 MTAG_PING	MUST be set to 0
0x00000005 MTAG_CONNECTION_REQ	MUST contain the ID of the connection being requested.
0x00000FFF MTAG_USER_MESSAGE	MUST contain the ID of the connection that the message relates to.

dwUserMsgType (4 bytes): A 4-byte integer value that contains additional details about the message, depending on the value of the **MsgTag** field, as follows.

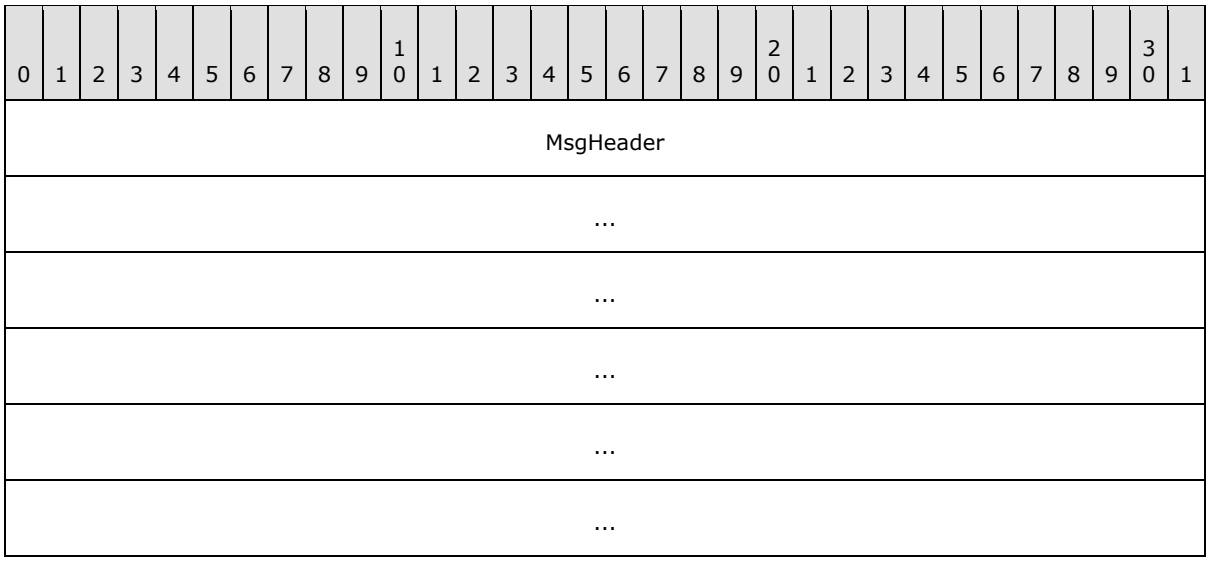
MsgTag field value	dwConnectionId field
0x00000001 MTAG_DISCONNECT	MUST contain the protocol type of the connection being disconnected.
0x00000002 MTAG_DISCONNECTED	MUST be set to 0.
0x00000003 MTAG_CONNECTION_REQ_DENIED	MUST be set to 0.
0x00000004 MTAG_PING	MUST be set to 0.
0x00000005 MTAG_CONNECTION_REQ	MUST contain the protocol type of the connection being requested.
0x00000FFF MTAG_USER_MESSAGE	MUST contain the type of user message to be delivered.

dwcbVarLenData (4 bytes): Unsigned 4-byte integer value that contains the size, in bytes, of the variable-length data buffer. This value MUST NOT be greater than 81880. This number is the maximum size of a boxcar, as specified in section 2.1.1.2, minus the size of a BOX_CAR_HEADER and the MESSAGE_PACKET itself, which is logically the largest single message that is possible to be transmitted in MSDTC Connection Manager: OleTx Multiplexing Protocol.

dwReserved1 (4 bytes): Reserved. This value MUST be set to an implementation-specific value, and MUST be ignored on receipt.<3>

2.2.3 MTAG_DISCONNECT

The MTAG_DISCONNECT message indicates a request to disconnect the specified connection.

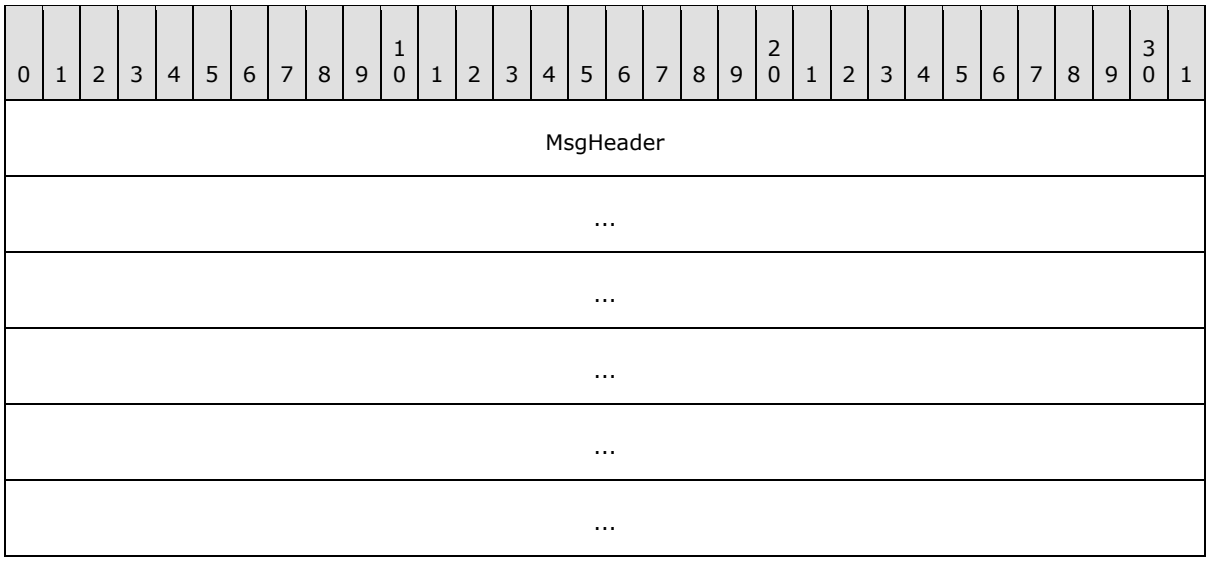


MsgHeader (24 bytes): This field contains a MESSAGE_PACKET structure. The fields MUST be set as specified in section 2.2.2. In particular:

- The **MsgTag** field MUST be set to 0x00000001 (MTAG_DISCONNECT).
- The **fIsMaster** field MUST be set to 0x00000001.
- The **dwcbVarLenData** field MUST be set to 0.

2.2.4 MTAG_DISCONNECTED

The MTAG_DISCONNECT message indicates that the request to disconnect the specified connection was successful.



MsgHeader (24 bytes): This field contains a MESSAGE_PACKET structure. The fields MUST be set as specified in section 2.2.2. In particular:

- The **MsgTag** field MUST be set to 0x00000002 (MTAG_DISCONNECTED).
- The **fIsMaster** field MUST be set to 0x00000000.
- The **dwcbVarLenData** field MUST be set to 0.

2.2.5 MTAG_CONNECTION_REQ_DENIED

The MTAG_CONNECTION_REQ_DENIED message indicates that the connection request for the specified connection has been denied. It represents a not-acknowledged response to an MTAG_CONNECTION_REQUEST message. (There is no positive acknowledgment response to an MTAG_CONNECTION_REQUEST message.)

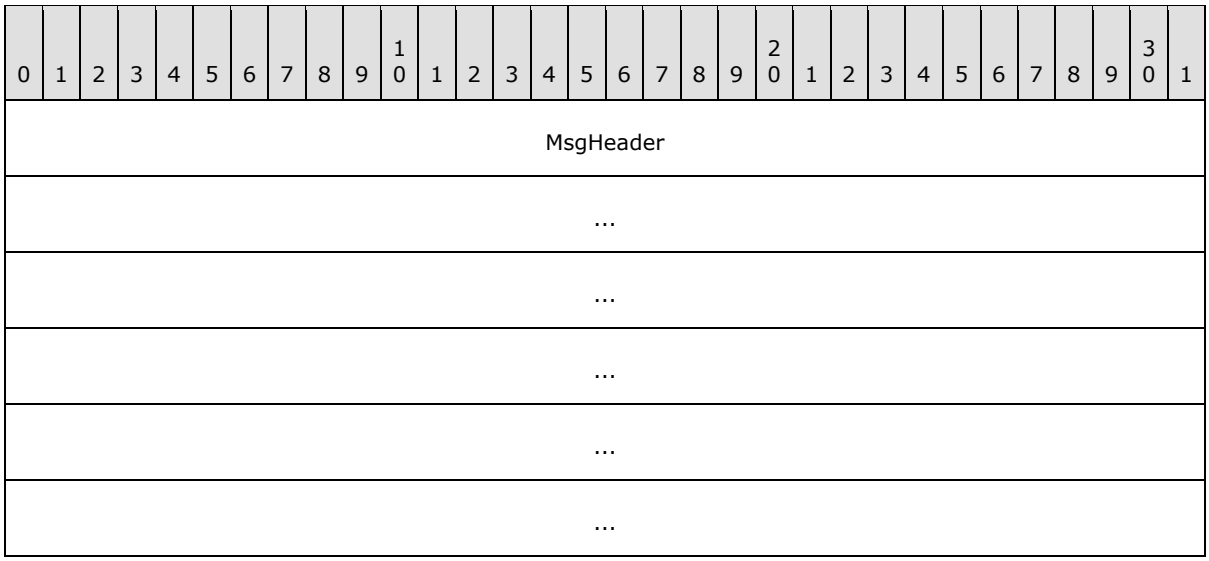
MsgHeader (24 bytes): This field contains a MESSAGE_PACKET structure. The fields MUST be set as specified in section 2.2.2. In particular:

- The **MsgTag** field MUST be set to 0x00000003 (MTAG_CONNECTION_REQ_DENIED).
- The **fIsMaster** field MUST be set to 0x00000000.
- The **dwcbVarLenData** field MUST be set to 4.

Reason (4 bytes): This field contains a 4-byte unsigned integer that indicates the reason that the connection request was denied. The values for this field are defined by the higher-layer protocol.

2.2.6 MTAG_PING

The MTAG_PING message is used by a protocol participant to determine if it can still contact its MSDTC Connection Manager: OleTx Transports Protocol session partner (for more information about the message processing event, see 3.1.5.4).

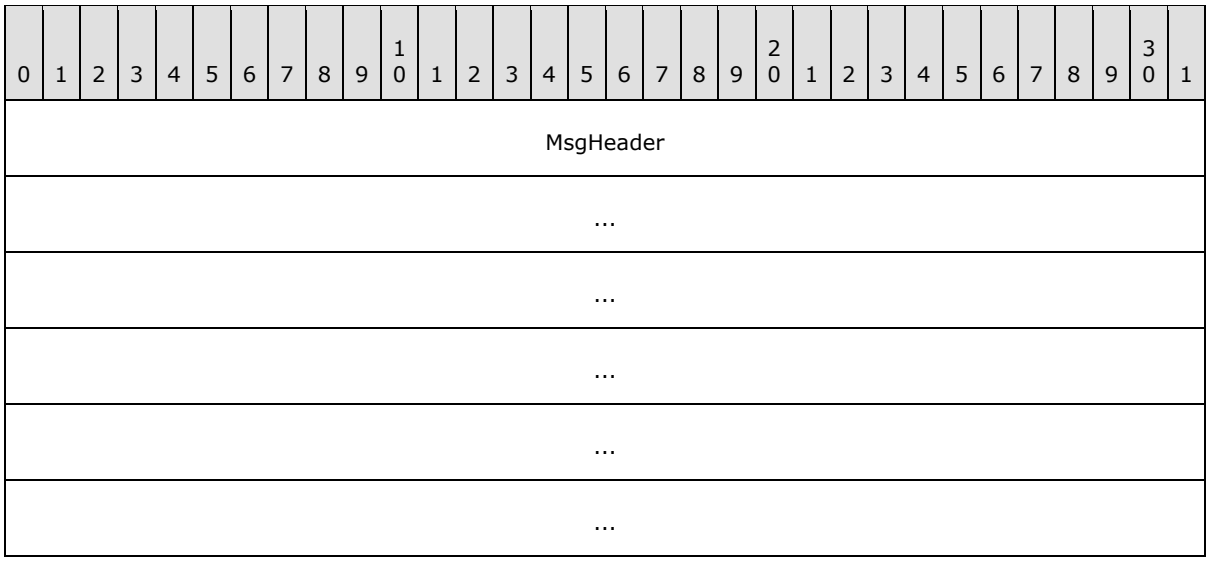


MsgHeader (24 bytes): This field contains a MESSAGE_PACKET structure. The fields MUST be set as specified in section 2.2.2. In particular:

- The **MsgTag** field MUST be set to 0x00000004 (MTAG_PING).
- The **fIsMaster** field MUST be set to 0x00000001.
- The **dwConnectionId** field MUST be set to 0.
- The **dwcbVarLenData** field MUST be set to 0.

2.2.7 MTAG_CONNECTION_REQ

The MTAG_CONNECTION_REQ message indicates a request to create the specified connection. A not-acknowledged response to this message is communicated with an MTAG_CONNECTION_REQ_DENIED message. (There is no positive acknowledgment response to an MTAG_CONNECTION_REQ message.)

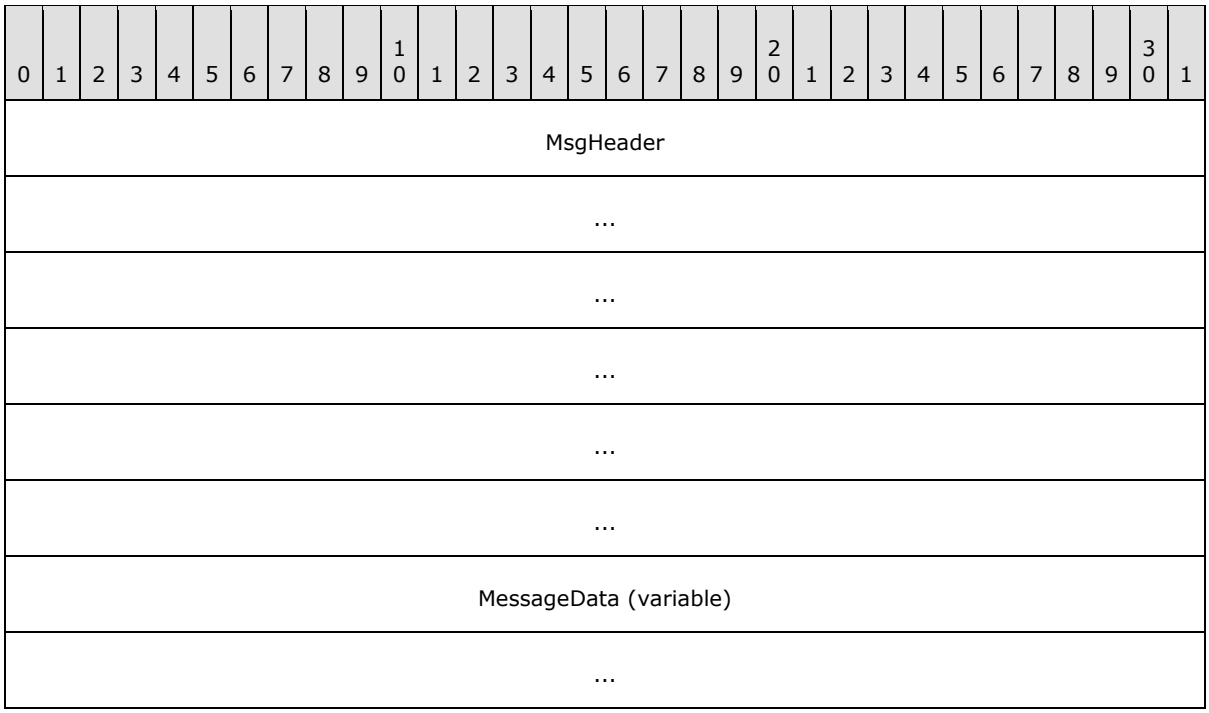


MsgHeader (24 bytes): This field contains a MESSAGE_PACKET structure. The fields MUST be set as specified in section 2.2.2. In particular:

- The **MsgTag** field MUST be set to 0x00000005 (MTAG_CONNECTION_REQ).
- The **fIsMaster** field MUST be set to 0x00000001.
- The **dwcbVarLenData** field MUST be set to 0.

2.2.8 MTAG_USER_MESSAGE

The MTAG_USER_MESSAGE message indicates that a user message should be delivered on the specified connection.



MsgHeader (24 bytes): This field contains a MESSAGE_PACKET structure. The fields MUST be set as specified in section 2.2.2. In particular:

- The **MsgTag** field MUST be set to 0x00000FFF (MTAG_USER_MESSAGE).
- The **dwcbVarLenData** field MUST be set to the length in bytes of the **MessageData** field, if it is present, otherwise it MUST be set to 0.

MessageData (variable): A byte array containing the body of the message. The format of this body is defined by the higher-layer software operating over this protocol, and it is generally indicated by the value of the **dwUserMsgType** field in the **MsgHeader** structure. The contents of this field MUST be treated as opaque.

3 Protocol Details

3.1 Common Details

3.1.1 Abstract Data Model

This section describes a conceptual model of a 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 performs. This document does not mandate that the implementations adhere to this model as long as their external behavior is consistent with that described in this document.

An MSDTC Connection Manager: OleTx Multiplexing Protocol implementation MUST maintain the following data elements:

Session Table: a table of Session objects, as maintained by an MSDTC Connection Manager: OleTx Transports Protocol partner and as specified in [MS-CMPO] section 3.1. The Session object MUST be extended to support the following additional data elements:

Outgoing Connection Table: a table of Connection objects indexed by the connection **ID** field of the Connection object.

Count of Allocated Outgoing Connections: an unsigned 32-bit integer counting the number of outgoing connections that have been allocated from the other partner in the session.

Incoming Connection Table: a table of Connection objects (section 3.1.1.1) indexed by the connection **ID** field of the Connection object.

Count of Allocated Incoming Connections: an unsigned 32-bit integer counting the number of outgoing connections that have been allocated by the other partner in the session.

Boxcar Queue: an ordered queue of Boxcar objects to be transmitted on this session.

Note that it is possible to implement the conceptual data by using a variety of techniques. An implementation is at liberty to implement such data in any way it pleases.

3.1.1.1 Connection Object

A Connection Object MUST contain the following data elements:

Connection ID: An unsigned 32-bit integer that identifies the connection. The Connection ID MUST be unique within a table. Note that a given Connection object is allowed to have the same Connection ID as another Connection object (related to the same Session object), so long as the other Communication object is in the other connection table. For example, the first connection is in the Incoming Connection Table and the second connection is in the Outgoing Connection Table, or vice versa).

Accepted: A Boolean value, indicating whether the connection was accepted or rejected by the higher-layer protocol. This value is initially false.

Protocol Type: An unsigned 32-bit integer that identifies the type of messages sent over the connection.

Note It is possible to implement the conceptual data by using a variety of techniques. An implementation is at liberty to implement such data in any way it pleases.

3.1.1.2 Boxcar Object

A Boxcar Object MUST contain the following data elements:

Message List: A list of MESSAGE_PACKET structures (section 2.2.2) in the boxcar.

When called for, Boxcar objects MUST be formatted and transmitted as specified in section 2.1.1.1.

Note It is possible to implement the conceptual data by using a variety of techniques. An implementation is at liberty to implement such data in any way it wants.

3.1.2 Timers

An MSDTC Connection Manager: OleTx Multiplexing Protocol implementation MUST maintain the following timers.

3.1.2.1 Idle Timer

There is an instance of the Idle Timer corresponding to each Session object. This timer MUST be set when both the Incoming Connection Table and the Outgoing Connection Table are empty, and it MUST be canceled when a Connection Object is added to either the Incoming Connection Table or the Outgoing Connection Table. The default value of the timer is specific to the implementation.<4>

3.1.3 Initialization

An MSDTC Connection Manager: OleTx Multiplexing Protocol instance is explicitly initialized with the following values required for the initialization of its underlying MSDTC Connection Manager: OleTx Transports Protocol instance, as specified in [MS-CMPO], section 3.2.3 Initialization:

- A local **Name Object**.
- The minimum and maximum level-three version numbers (as specified in [MS-CMPO], section 3.2.1.1 Local Partner State).
- A security level.

When the instance is initialized, an underlying MSDTC Connection Manager: OleTx Transports Protocol partner is initialized with the local Name Object and the provided level 3 version numbers (as specified in [MS-CMPO], section 3.2.1.1 Local Partner State). Both of the level 2 version numbers that are used to initialize the underlying MSDTC Connection Manager: OleTx Transports Protocol partner MUST be 1 (as specified in [MS-CMPO], section 3.2.3 Initialization).

3.1.4 Higher-Layer Triggered Events

3.1.4.1 Send Message

When the higher-layer protocol requests to send a message, it MUST specify the Connection object on which to send the message (which implies the connection table containing it), an unsigned 32-bit integer representing the type of message, and a byte array containing the body of the message. The byte array MUST NOT be more than 81880 bytes long.

The MSDTC Connection Manager: OleTx Multiplexing Protocol instance MUST allocate an MTAG_USER_MESSAGE message. It MUST set the **dwUserMsgType** field in the **MsgHeader** field to the provided message type, it MUST set the **dwConnectionId** field in the **MsgHeader** field to the Connection ID of the provided Connection object, it MUST set the dwcbVarLenData field in the **MsgHeader** field to the length of the provided array, and it MUST set the **MessageData** field to the

provided byte array. Finally, if the provided Connection object is contained in an Outgoing Connection Table, then the **fIsMaster** field of the **MsgHeader** field MUST be set to 0x00000001; otherwise, it MUST be set to 0x00000000.

This message MUST be enqueued on the Session object associated with the provided Connection object as described in section 3.1.7.1.

3.1.4.2 Create Connection

When the higher-layer protocol requests a new connection, it MUST specify the Name Object of the partner to create the connection with and the protocol type of the connection to create.

First, the MSDTC Connection Manager: OleTx Multiplexing Protocol instance MUST look up the Session object with the specified Name Object in the Session Table. If a matching session does not exist, the MSDTC Connection Manager: OleTx Multiplexing Protocol instance MUST request a new session with the partner from the underlying MSDTC Connection Manager: OleTx Transports Protocol instance. If the request is unsuccessful, then the connection request MUST fail. The MSDTC Connection Manager: OleTx Multiplexing Protocol instance MUST initialize its extensions to the Session object as follows:

- The Incoming Connection Table MUST be empty.
- The Outgoing Connection Table MUST be empty.
- The Count of Allocated Outgoing Connections MUST be zero.
- The Count of Allocated Incoming Connections MUST be zero.
- The Boxcar Queue MUST be empty.

After a Session object has been found or created, the MSDTC Connection Manager: OleTx Multiplexing Protocol instance MUST compare the number of Connection objects in the Outgoing Connection Table in the Session object with the Count of Allocated Outgoing Connections. If they are equal, then the MSDTC Connection Manager: OleTx Multiplexing Protocol instance MUST request resource allocation from the underlying MSDTC Connection Manager: OleTx Transports Protocol instance. The MSDTC Connection Manager: OleTx Multiplexing Protocol instance MUST provide the Session object. In addition, it MUST specify the type of resource as 0x00000000, and it MUST specify a quantity of at least 1. It MAY specify a quantity greater than 1.<5>

If the request is successful, then the number of resources that were actually allocated MUST be added to the Count of Allocated Outgoing Connections. Otherwise, the connection request MUST fail and the Session object MUST be left unmodified, with the exception that if both the Incoming and Outgoing Connection Tables are empty, then the Idle Timer associated with the Session object MUST be started.

Next, the MSDTC Connection Manager: OleTx Multiplexing Protocol instance MUST allocate a new Connection object with the specified protocol type and with a connection identifier that is currently unused in the Outgoing Connection Table. The **Accepted** field of the new Connection object MUST be set to true. This Connection Object MUST be added to the Outgoing Connection Table. If the Idle Timer is active, the timer MUST be canceled.

Finally, the MSDTC Connection Manager: OleTx Multiplexing Protocol instance MUST allocate an MTAG_CONNECTION_REQ message. It MUST set the **dwUserMsgType** field in the **MsgHeader** field to the specified protocol type, and it MUST set the **dwConnectionId** field in the **MsgHeader** field to the connection identifier of the new Connection object. It MUST enqueue the message on the Session object as described in section 3.1.7.1.

3.1.4.3 Disconnect Connection

When the higher-layer protocol requests to disconnect a connection, it MUST specify the Connection object to disconnect. This Connection object MUST be contained in an Outgoing Connection Table on a Session object contained in the Session Table; otherwise, the request to disconnect the connection MUST fail.

The MSDTC Connection Manager: OleTx Multiplexing Protocol instance MUST allocate an MTAG_DISCONNECT message and set the **dwConnectionId** field in the **MsgHeader** field of the message to the connection identifier of that specified connection object. It MUST enqueue this message on the Session object of the specified connection object as described in section 3.1.7.1.

3.1.5 Message Processing Events and Sequencing Rules

MSDTC Connection Manager: OleTx Multiplexing Protocol messages are received from the underlying MSDTC Connection Manager: OleTx Transports Protocol as specified in [MS-CMPO] section 3.1.4.4. The buffers that the MSDTC Connection Manager: OleTx Transports Protocol provides MUST be formatted boxcars as specified in [MS-CMPO] section 2.1.1.

The relative position of each message in the boxcar MUST be used to order the messages; messages that occur at a smaller offset from the boxcar header in the boxcar MUST be considered to come before messages that occur later in the boxcar. Boxcars MUST be ordered by the time of their receipt by an MSDTC Connection Manager: OleTx Multiplexing Protocol implementation; all of the messages in a boxcar that is received earlier than another boxcar are considered to come before all of the messages in the later boxcar. The **dwConnectionId** field of the message MUST be used to logically group messages; messages MUST be in the same group if their **dwConnectionId** fields are equal. (The messages MAY NOT be actually grouped by their **dwConnectionId** fields in the boxcar.)<6>

An MSDTC Connection Manager: OleTx Multiplexing Protocol implementation MUST NOT process a message until it has processed all messages in the same group that come before it. (Message order MUST be preserved within an [MS-CMP] connection.) An MSDTC Connection Manager: OleTx Multiplexing Protocol implementation MAY process messages in any order that does not violate the preceding restriction.<7>

All MSDTC Connection Manager: OleTx Multiplexing Protocol messages are extensions of the MESSAGE_PACKET structure as specified in section 2.2.2. An MSDTC Connection Manager: OleTx Multiplexing Protocol message is identified by looking at the value of the **MsgTag** field; the interpretation of the message depends on the value of that field. If the value of the **MsgTag** field is outside of the expected range (as specified in section 2.2.2), then all remaining unprocessed messages in the boxcar MAY be ignored, regardless of which connection they are intended for.<8>

3.1.5.1 MTAG_DISCONNECT (MsgTag 0x00000001)

When an MTAG_DISCONNECT message is received on a session, the MSDTC Connection Manager: OleTx Multiplexing Protocol implementation MUST look at the **dwConnectionId** field of the **MsgHeader** field of the message, and retrieve the Connection object with the matching connection ID from the Incoming Connection Table of the Session object. If no such Connection object exists, the MTAG_DISCONNECT message MUST be silently ignored.

Otherwise, the higher-layer protocol MUST be notified of the disconnect event, and the MSDTC Connection Manager: OleTx Multiplexing Protocol implementation MUST remove the connection object from the Incoming Connection Table of the Session object. If both the Incoming Connection Table and the Outgoing Connection Table of the Session object are now empty, the Idle Timer MUST be started.

The MSDTC Connection Manager: OleTx Multiplexing Protocol implementation MUST then allocate a new MTAG_DISCONNECTED message, set the **dwUserMsgType** field of the **MsgHeader** field to the protocol type of the Connection object, and set the **dwConnectionId** field of the **MsgHeader** to the connection ID of the Connection object. Finally, the message MUST be enqueued on the Session object as specified in section 3.1.7.1.

3.1.5.2 MTAG_DISCONNECTED (MsgTag 0x00000002)

When an MTAG_DISCONNECTED message is received on a session, the MSDTC Connection Manager: OleTx Multiplexing Protocol implementation MUST look at the **dwConnectionId** field of the **MsgHeader** field of the message, and retrieve the Connection object with the matching connection ID from the Outgoing Connection Table of the Session object. If no such Connection object exists, the MTAG_DISCONNECTED message MUST be silently ignored.

Otherwise, the higher-layer protocol MUST be notified of the connection disconnected event, and the MSDTC Connection Manager: OleTx Multiplexing Protocol implementation MUST remove the Connection object from the Outgoing Connection Table of the Session object. If there are no more connections in the Outgoing Connection Table of the Session object and there are no connections in the Incoming Connection Table of the Session object, then the Idle Timer MUST be started.

3.1.5.3 MTAG_CONNECTION_REQ_DENIED (MsgTag 0x00000003)

When an MTAG_CONNECTION_REQ_DENIED message is received on a session, the MSDTC Connection Manager: OleTx Multiplexing Protocol implementation MUST look at the **dwConnectionId** field of the **MsgHeader** field of the message, and retrieve the Connection object with the matching connection ID from the Outgoing Connection Table of the Session object. If no such Connection object exists, the MTAG_CONNECTION_REQ_DENIED message MUST be silently ignored.

Otherwise, the higher-layer protocol MUST be notified of the fact that the connection request was denied for the particular Connection object, along with the value in the **Reason** field of the message.

3.1.5.4 MTAG_PING (MsgTag 0x00000004)

A protocol implementation MAY send out MTAG_PING messages periodically to verify that its connection with a communication partner is active. (If the connection is unavailable, sending the MTAG_PING message will return an error.) When an MTAG_PING message is received on a session, the MSDTC Connection Manager: OleTx Multiplexing Protocol implementation MUST ignore it.<9>

3.1.5.5 MTAG_CONNECTION_REQ (MsgTag 0x00000005)

When an MTAG_CONNECTION_REQ message is received on a Session object, the MSDTC Connection Manager: OleTx Multiplexing Protocol implementation MUST first compare the number of Connection objects in the Incoming Connection Table on the Session object with the Count of Allocated Incoming Connections on the Session object. If the Count of Allocated Incoming Connections is equal to the number of Connection objects in the table, then the MSDTC Connection Manager: OleTx Multiplexing Protocol implementation MUST ignore the MTAG_CONNECTION_REQ message.

Otherwise, the MSDTC Connection Manager: OleTx Multiplexing Protocol implementation MUST look at the **dwConnectionId** field of the **MsgHeader** field of the message, and attempt to retrieve the Connection object with the matching connection ID from the Incoming Connection Table of the Session object. If a Connection object is found, then the MTAG_CONNECTION_REQ message MUST be silently ignored.

Otherwise, the MSDTC Connection Manager: OleTx Multiplexing Protocol implementation MUST allocate a Connection object, initializing the protocol type field to the **dwUserMsgType** field of the **MsgHeader** field of the message, the **Accepted** field to false, and the connection **ID** field to the **dwConnectionId** field of the **MsgHeader** field of the message. It MUST add the Connection object to the Incoming Connection Table of the Session object. If the Idle Timer is active, then it MUST be canceled.

The implementation MUST then notify the higher-layer protocol of the incoming connection, providing the Connection object and its protocol type. The higher-layer protocol MUST either accept or reject the connection.

If the higher-layer protocol rejects the connection, then it MUST provide a protocol-specific, 32-bit unsigned integer that specifies the reason for the rejection. The implementation MUST then allocate a new MTAG_CONNECTION_REQ_DENIED message, initializing the **dwConnectionId** field of the **MsgHeader** field to the connection ID of the Connection object and the **Reason** field to the unsigned integer provided by the higher-layer protocol. It MUST then enqueue this message on the Session object as specified in section 3.1.7.1.

If the higher-layer protocol accepts the connection, then the MSDTC Connection Manager: OleTx Multiplexing Protocol implementation MUST set the **Accepted** field of the Connection object to true.

3.1.5.6 MTAG_USER_MESSAGE (MsgTag 0x00000FFF)

When an MTAG_USER_MESSAGE message is received on a Session object, the MSDTC Connection Manager: OleTx Multiplexing Protocol implementation MUST examine the **fIsMaster** field of the **MsgHeader** field of the message to determine which table contains the destination Connection object. If the **fIsMaster** field is 0x00000000, then the MSDTC Connection Manager: OleTx Multiplexing Protocol implementation MUST attempt to find a Connection object with a connection ID that matches the **dwConnectionId** field of the **MsgHeader** field of the message in the Incoming Connection Table of the Session object. Otherwise, the MSDTC Connection Manager: OleTx Multiplexing Protocol implementation MUST attempt to find a Connection object with a connection ID that matches the **dwConnectionId** field of the **MsgHeader** field of the message in the Outgoing Connection Table of the Session object.

If no Connection object is found in the selected table, or the **Accepted** field of the Connection object is false, then the MSDTC Connection Manager: OleTx Multiplexing Protocol implementation MUST ignore the message. Otherwise, the higher-layer protocol MUST be notified of the incoming message. The MSDTC Connection Manager: OleTx Multiplexing Protocol implementation MUST provide the higher-layer protocol with the Connection object, the value of the **dwUserMsgType** field of the **MsgHeader** field of the message, and the **MessageData** field of the MTAG_USER_MESSAGE field if it is present.

3.1.6 Timer Events

3.1.6.1 Idle Timer

When the Idle Timer associated with a Session object expires, an implementation of the MSDTC Connection Manager: OleTx Multiplexing Protocol MUST request a forced session teardown for the underlying MSDTC Connection Manager: OleTx Transports Protocol Session object. Note that the Idle Timer is only active when there are no Connection objects in either the Outgoing Connection Table or the Incoming Connection Table; therefore, there is no need to inform the higher-layer protocol of the teardown.

3.1.7 Other Local Events

3.1.7.1 Enqueuing a Message

Various events in the protocol require that a message be queued on a particular Session object. This section describes how this is done.

If it is possible to add the provided message to the end of the Message List of the last Boxcar object, in the Boxcar Queue associated with the provided Session object, then it MUST be added to that Boxcar object. (The constraints governing whether it is possible to add a message to the list are provided in section 2.1.1.2.) Otherwise, a new Boxcar object MUST be allocated and added to the end of the Boxcar Queue associated with the provided Session object; the message MUST then be added to the end of the Message List in new Boxcar object instead.

An MSDTC Connection Manager: OleTx Multiplexing Protocol implementation MAY choose to transmit the Boxcar object at the head of the Boxcar Queue at any time, so long as it contains at least one message; however, an implementation SHOULD transmit this Boxcar as soon as possible when there is at least one other Boxcar object in the Boxcar Queue. Boxcars MUST be formatted and transmitted as described in section 2.1.1.1.<10>

3.1.7.2 Session Down

When the underlying MSDTC Connection Manager: OleTx Transports Protocol Session object is torn down or fails for any reason other than the expiration of the Idle Timer, the higher-layer protocol MUST be notified of the teardown. The higher-layer protocol MUST be provided with every Connection object in both the Outgoing Connection Table and the Incoming Connection Table of the Session object. The Connection objects MUST then be removed from their containing tables. Any resources associated with the session MAY also be reclaimed at this time.<11>

4 Protocol Examples

In the following examples, there are two MSDTC Connection Manager: OleTx Transaction Protocol Specification instances; **initiator** and **acceptor**. It is assumed that the two instances have established a session with each other, and that the initiator has negotiated a sufficient number of resources with the acceptor.

- Sending Messages
- A Simple Connection Scenario

4.1 Sending Messages

The Sending Messages example shows how the initiator creates the appropriate structures to create a connection and then sends a message on that connection. In this case, the protocol type of the connection is 0x00000101, and the user message type of the first message is 0x00002001. (These values are CONNTYPE_PARTNERTM_PROPAGATE and PARTNERTM_PROPAGATE_MTAG_PROPAGATE, respectively, as specified in [MS-DTCO].)

The initiator is going to create two MESSAGE_PACKET structures, format them into a boxcar, and then submit them to the underlying MSDTC Connection Manager: OleTx Transports Protocol session to be transmitted. (Because it is assumed that a connection request will succeed, both MESSAGE_PACKET structures are put into the same boxcar.)

4.1.1 Creating the MESSAGE_PACKETs

To start the connection, the initiator allocates the next free connection identifier; in this instance, it is 0x00000001. The initiator then creates a MESSAGE_PACKET, with the **MsgTag** field set to MTAG_CONNECTION_REQ (0x00000005) and with the **dwUserMsgType** field set to 0x00000101 (which is CONNTYPE_PARTNERTM_PROPAGATE as specified in [MS-DTCO]). By definition, the instance that creates the connection always sets the **fIsMaster** field to 1, and as this MESSAGE_PACKET structure contains no extra data, the **dwcbVarLenData** is set to 0.

The following table displays the first MESSAGE_PACKET structure that the initiator creates (all values are 32-bits wide).

Field	Value	Value description
MsgTag	0x00000005	MTAG_CONNECTION_REQ
fIsMaster	0x00000001	1
dwConnectionId	0x00000001	1
dwUserMsgType	0x00000101	CONNTYPE_PARTNERTM_PROPAGATE
dwcbVarLenData	0x00000000	0
dwReserved1	0xcd64cd64	dwReserved1: 0xcd64cd64

The initiator then creates a second MESSAGE_PACKET to contain the user message. It sets the **MsgTag** field to MTAG_USER_MESSAGE (0x00000fff) and the **dwUserMsgType** field to 0x00002001 (which is PARTNERTM_PROPAGATE_MTAG_PROPAGATE as specified in [MS-DTCO]).

The MESSAGE_PACKET also contains an extra 60 bytes of data for the message body, so it sets the **dwcbVarLenData** field to 0x0000003c. The message body that follows is specific to the message;

in this instance, it specifies a transaction ID (**guidTx**, set to 9fa8a337-eaf7-4230-9232-b57379d65077), a transaction isolation level (**isoLevel**, set to 0x00100000, which is ISOLATIONLEVEL_SERIALIZABLE), and a transaction description (**szDesc**, set to the string "Example Transaction - 39 chars long...."). The following table is the second MESSAGE_PACKET structure that the initiator creates (again, all of the values are 32 bits wide).

Field	Value	Value Description
MsgTag	0x00000fff	MTAG_USER_MESSAGE
fIsMaster	0x00000001	1
dwConnectionId	0x00000001	1
dwUserMsgType	0x00002001	PARTNERTM_PROPAGATE_MTAG_PROPAGATE
dwcbVarLenData	0x0000003c	60
dwReserved1	0xcd64cd64	dwReserved1: 0xcd64cd64
guidTx	0x9fa8a337 0x4230eaf7 0x73b53292 0x7750d679	9fa8a337-eaf7-4230-9232-b57379d65077
isoLevel	0x00100000	ISOLATIONLEVEL_SERIALIZABLE
szDesc	0x6d617845 0x20656c70 0x6e617254 0x74636173 0x206e6f69 0x3933202d 0x61686320 0x6c207372 0x2e676e6f 0x002e2e2e	"Example Transaction - 39 chars long...."

To send these MESSAGE_PACKETs, the initiator wraps the two messages into a single boxcar, which is in turn passed to the underlying MSDTC Connection Manager: OleTx Transports Protocol instance.

4.1.2 Creating a Boxcar

A boxcar always begins with a BOX_CAR_HEADER structure. The first two fields (**dwSeqNumThisCar** and **dwAckSeqNum**) are reserved and are always set to zero. The third field (**dwcbTotal**) contains the total number of bytes in the Boxcar (in this case, 0x00000080; 128 bytes.) The fourth field (**dwcMessages**) contains the total number of MESSAGE_PACKETs in the BOX_CAR_HEADER (in this case, 2).

The rest of the boxcar contains an array of MESSAGE_PACKET structures. In this example, the two MESSAGE_PACKET structures from section 4.1.1 are included in this boxcar. Note that individual MESSAGE_PACKET structures are aligned to 8-byte boundaries, and that they are present in the order that they are intended to be processed. The following is the final boxcar structure.

Field	Value	Value description
dwSeqNumThisCar	0x00000000	dwSeqNumThisCar: 0
dwAckSeqNum	0x00000000	dwAckSeqNum: 0
dwcbTotal	0x00000080	dwcbTotal: 128
dwcMessages	0x00000002	dwcMessages: 2
MsgTag	0x00000005	MTAG_CONNECTION_REQ
fIsMaster	0x00000001	1
dwConnectionId	0x00000001	1
dwUserMsgType	0x00000101	CONNTYPE_PARTNERTM_PROPAGATE
dwcbVarLenData	0x00000000	0
dwReserved1	0xcd64cd64	dwReserved1: 0xcd64cd64
MsgTag	0x00000FFF	MTAG_USER_MESSAGE
fIsMaster	0x00000001	1
dwConnectionId	0x00000001	1
dwUserMsgType	0x00002001	PARTNERTM_PROPAGATE_MTAG_PROPAGATE
dwcbVarLenData	0x00000040	64
dwReserved1	0xcd64cd64	dwReserved1: 0xcd64cd64
guidTx	0x9fa8a337 0x4230eaf7 0x73b53292 0x7750d679	9fa8a337-eaf7-4230-9232-b57379d65077
isoLevel	0x00100000	ISOLATIONLEVEL_SERIALIZABLE
szDesc	0x6d617845 0x20656c70 0x6e617254 0x74636173 0x206e6f69 0x3933202d 0x61686320 0x6c207372 0x2e676e6f 0x002e2e2e 0x00000000	"Example Transaction - 39 chars long...." Padding

4.1.3 Sending the Boxcar Using the Underlying MSDTC Connection Manager: OleTx Transports Protocol Session

Now that the boxcar has been constructed, the initiator submits the boxcar to the underlying MSDTC Connection Manager: OleTx Transports Protocol session for transmission. The initiator simply hands the buffer above to the MSDTC Connection Manager: OleTx Transports Protocol session, along with the count of messages that it contains (2). The MSDTC Connection Manager: OleTx Transports Protocol session will ensure that the boxcar is delivered to the acceptor, which will parse it and process the messages it contains.

4.2 A Simple Connection Scenario

In this example, the initiator starts a connection and then, when all of the messages associated with the connection are complete, the initiator disconnects the connection.

4.2.1 Initiating a Connection

Sending Messages shows how the initiator would create two MESSAGE_PACKET structures to request a new connection with protocol type 0x00002001 (CONNTYPE_PARTNERTM_PROPAGATE as specified in [MS-DTCO]) and send the first message. In this scenario, after the first message is sent, the protocol type that the initiator has requested indicates that the initiator should wait for some sort of response message. As this will be the first message that the initiator receives on the connection, the initiator will also be informed that the connection request was denied.

4.2.1.1 Connection Denied

Assume for a moment that the acceptor denies the connection request, then the acceptor will create a MESSAGE_PACKET with the **MsgTag** field set to MTAG_CONNECTION_REQ_DENIED (0x00000003), and it will provide a reason for the rejection in the **Reason** field (for example, E_ACCESSDENIED, or 0x80070005), which is appended to the end of the MESSAGE_PACKET. It will set the **dwConnectionId** field to the connection identifier that the initiator requested (0x00000001), and it will set the **dwcbVarLenData** field to four (the size of the **Reason** field that follows). The **dwUserMsgType** field is set to zero, because this is a MTAG_CONNECTION_REQ_DENIED message; likewise, the **fIsMaster** field is set to 0. The acceptor will then drop all incoming messages with a **dwConnectionId** field set to 1 until it receives a disconnect request.

The MESSAGE_PACKET structure is as follows.

Field	Value	Description
MsgTag	0x00000003	MTAG_CONNECTION_REQ_DENIED
fIsMaster	0x00000000	0
dwConnectionId	0x00000001	1
dwUserMsgType	0x00000000	dwUserMsgType: 0
dwcbVarLenData	0x00000004	4
dwReserved1	0xcd64cd64	dwReserved1: 0xcd64cd64
dwReason	0x80070005	E_ACCESSDENIED

4.2.1.2 Connection Accepted

If the acceptor accepts the connection request instead, then it does not send back a specific message to that effect. Instead, the acceptor will move on to process the next message in the boxcar. In this case, the next message is a user message, and so it is passed to the higher-layer protocol (in this case, MSDTC Connection Manager: OleTx Transaction Protocol) for processing.

In this example, the higher-layer protocol will respond with another user message. The acceptor will create a MESSAGE_PACKET structure with the **MsgTag** field set to MTAG_USER_MESSAGE (0x00000FFF), and the **dwUserMsgType** field set to 0x00002002 (which is PARTNERTM_PROPAGATE_MTAG_PROPAGATED, the message that MSDTC Connection Manager: OleTx Transaction Protocol sends to indicate that the PARTNERTM_PROPAGATE_MTAG_PROPAGATED message was processed successfully.) The message has no body, so the **dwcbVarLenData** field is set to 0, and the message is being sent by the acceptor, so the **fIsMaster** field is set to 0. The message is being sent as a response on the connection that the initiator started, so the **dwConnectionId** field is set to 1.

The response MESSAGE_PACKET ultimately looks like the following.

Field	Value	Description
MsgTag	0x00000FFF	MTAG_USER_MESSAGE
fIsMaster	0x00000000	0
dwConnectionId	0x00000001	1
dwUserMsgType	0x00002002	PARTNERTM_PROPAGATE_MTAG_PROPAGATED
dwcbVarLenData	0x00000000	0
dwReserved1	0xcd64cd64	dwReserved1: 0xcd64cd64

Regardless of whether the acceptor chooses to accept or reject the connection, the MESSAGE_PACKET that the acceptor generates is packed into a boxcar (as described earlier) and it is transmitted back to the initiator.

4.2.2 Disconnecting a Connection

The initiator is responsible for disconnecting the connection when the connection is complete, even if the connection was denied by the acceptor.

The initiator begins the disconnect sequence for a connection by creating a MESSAGE_PACKET structure with the **MsgTag** field set to MTAG_DISCONNECT (0x00000001), the **dwConnectionId** field set to the identifier of the connection being disconnected (0x00000001), and the **dwUserMsgType** field set to zero.

The MESSAGE_PACKET structure is as follows.

Field	Value	Description
MsgTag	0x00000001	MTAG_DISCONNECT
fIsMaster	0x00000001	1
dwConnectionId	0x00000001	1

Field	Value	Description
dwUserMsgType	0x00000000	dwUserMsgType: 0
dwcbVarLenData	0x00000000	0
dwReserved1	0xcd64cd64	dwReserved1: 0xcd64cd64

The initiator packages this MESSAGE_PACKET into a boxcar and sends it to the acceptor over the underlying MSDTC Connection Manager: OleTx Transports Protocol session.

When the acceptor receives the disconnect request, the acceptor begins the process of cleaning up any connection-specific resources. After this process is complete, the acceptor creates a MESSAGE_PACKET structure with the **MsgTag** field set to MTAG_DISCONNECTED (0x00000002), the **dwConnectionId** field set to the identifier of the connection that was just disconnected (0x00000001), and the **dwUserMsgType** field set to zero (0). The complete MESSAGE_PACKET structure is as follows:

Field	Value	Description
MsgTag	0x00000002	MTAG_DISCONNECTED
fIsMaster	0x00000000	0
dwConnectionId	0x00000001	1
dwUserMsgType	0x00000000	dwUserMsgType: 0
dwcbVarLenData	0x00000000	0
dwReserved1	0xcd64cd64	dwReserved1: 0xcd64cd64

When the initiator receives the disconnected message, the initiator then cleans up any connection-specific resources and reclaims the connection identifier for future use.

5 Security

The following sections specify security considerations for implementers of the MSDTC Connection Manager: OleTx Multiplexing Protocol.

5.1 Security Considerations for Implementers

This protocol has no additional security considerations beyond those discussed in [MS-CMPO], section 5.1 Security Considerations for Implementers.

5.2 Index of Security Parameters

None.

6 Appendix A: Windows Behavior

The information in this specification is applicable to the following versions of Windows:

- Windows NT 4.0 Option Pack
- Windows 2000
- Windows XP
- Windows Server 2003
- Windows Vista
- Windows Server 2008

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 Windows behavior in accordance with the SHOULD or SHOULD NOT prescription. Unless otherwise specified, the term MAY implies that Windows does not follow the prescription.

<1> Section 2.1.1.2: Windows always adds a message to an existing boxcar unless doing so would cause the boxcar to exceed any of the limits set out in section 2.1.1.2.

<2> Section 2.2: Windows does not reliably set the padding bytes to zero; instead, the bytes are filled with garbage.

<3> Section 2.2.2: All versions of Windows set this field to a random 4-byte value.

<4> Section 3.1.2.1: For all versions of Windows, the default value of this timer is 10 minutes.

<5> Section 3.1.4.2: All versions of Windows periodically negotiate for the number of connections that it needs by maintaining a queue of connections that are waiting to connect but have not yet been negotiated. A timer with a default period of 6 seconds is used to regulate how often to negotiate the resources, and connection requests are blocked until the timer expires. However, the higher-layer protocol may make a special connection request such that all of the connections in the queue have their resources negotiated as soon as possible.

<6> Section 3.1.5: Windows does not physically group the messages by **dwConnectionId** in the boxcar.

<7> Section 3.1.5: Windows processes messages in the order in which they appear in the boxcar, and does not otherwise reorder messages.

<8> Section 3.1.5: As Windows processes the messages strictly in the order in which they appear in the boxcar, it discards the remainder of the boxcar when it encounters such a message.

<9> Section 3.1.5.4: All versions of Windows transmit these messages at regular intervals when there are no other messages to be sent; by default, they are sent every six seconds while the Idle Timer (as specified in 3.1.2.1) is active, although this is configurable.

<10> Section 3.1.7.1: Windows uses several different mechanisms for determining when to transmit a Boxcar:

- When a Boxcar is full, it is transmitted immediately (pending the completion of any outstanding transmission.)

- When a higher-layer protocol requests a "rush" send, the boxcar that the message is placed in is transmitted pending the completion of any outstanding transmission, and the transmission of every boxcar ahead of it in the queue.
- At regular intervals, Windows transmits the boxcar at the head of the queue. If there is no boxcar, or the boxcar is empty, Windows will put an MTAG_PING message into it before transmitting it.

<11> Section 3.1.7.2: Windows reclaims all resources associated with the Session object at this time.

7 Index

P

R

S

T

V

W