[MC-SMP]: Session Multiplex 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
08/10/2007	0.1	Major	Initial Availability
09/28/2007	0.2	Minor	Updated the technical content.
10/23/2007	0.2.1	Editorial	Revised and edited the technical content.
11/30/2007	0.2.2	Editorial	Revised and edited the technical content.
01/25/2008	0.2.3	Editorial	Revised and edited the technical content.

Table of Contents

1 Introduction 4
1.1 Glossary 4
1.2 References 4
1.2.1 Normative References 4
1.2.2 Informative References 5
1.3 Protocol Overview (Synopsis) 5
1.4 Relationship to Other Protocols 6
1.5 Prerequisites/Preconditions 7
1.6 Applicability Statement 7
1.7 Versioning and Capability Negotiation 7
1.8 Vendor-Extensible Fields 7
1.9 Standards Assignments 7

2 Messages 8
2.1 Transport 8
2.2 Message Syntax 8
2.2.1 Header 8
2.2.1.1 Control Flags 9
2.2.2 SYN Packet 9
2.2.3 ACK Packet 10
2.2.4 FIN Packet 11
2.2.5 DATA Packet 12

3 Protocol Details 14
3.1 Common Details 14
3.1.1 Abstract Data Model 16
3.1.1.1 Per Transport Connection Structure 16
3.1.1.2 Per Session Structure 16
3.1.1.3 Flow Control Algorithm 17
3.1.1.3.1 Sender 17
3.1.1.3.2 Receiver 18
3.1.1.3.3 Update Sender's HighWaterforSend Variable Using a ACK Packet 18
3.1.2 Timers 19
3.1.3 Initialization 19
3.1.3.1 Per Transport Connection Structure 19
3.1.3.2 Per Session Structure 19
3.1.4 Higher-Layer Triggered Events 19
3.1.4.1 Initiating the Session Multiplex Protocol on a Transport Endpoint 19
3.1.4.2 Shutting Down the Session Multiplex Protocol on a Transport Endpoint 19
3.1.4.3 Initiating Session Establishment (Open Call) 20
3.1.4.4 Sending Messages to Peer and Receiving Messages from Peer (Send and Receive Call) 20
3.1.4.5 Session Termination (Close Call) 21
3.1.5 Message Processing Events and Sequencing Rules 21
3.1.5.1 Receiving a Packet 21
3.1.5.1.1 Receiving a SYN packet 21
3.1.5.1.2 Receiving a DATA packet 22
3.1.5.1.3 Receiving a ACK packet 22
3.1.5.1.4 Receiving a FIN packet 22
3.1.6 Timer Events 22
3.1.7 Other Local Events 22

4 Protocol Examples 23

4.1 New Session 23
4.2 Update Window - ACK 23
4.3 First Command on a Session 24
4.4 Last Command on a Session 24

5 Security 26
5.1 Security Considerations for Implementers 26
5.2 Index of Security Parameters 26

6 Appendix A: Windows Behavior 27

7 Index 28

1 Introduction

This document specifies the Session Multiplex Protocol, which is a Microsoft proprietary protocol. The Session Multiplex Protocol is an application-layer protocol that provides session management capabilities between a database client and a database server. Specifically, the Session Multiplex Protocol enables multiple logical client connections to connect to a single server over a single physical connection. This protocol can be used to support features such as Multiple Active Result Sets (MARS) (for more information, see [MSDN-MARS]).

1.1 Glossary

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

Client
Little-Endian
Server

The following terms are specific to this document:

Insert: To add a row to a table.

Multiple Active Result Sets (MARS): A feature introduced in SQL Server 2005 that allows applications to have more than one pending request per connection. Further information may be obtained from the SQL Server 2005 Books Online.

Tabular Data Stream (TDS): An application level protocol used to facilitate requests and responses between a database **server** and **client**. This is the protocol used by Microsoft SQL Server 2005.

Virtual Interface Architecture (VIA): A high-speed interconnect requiring special hardware and drivers provided by third parties.

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.

[IEEE754] Institute of Electrical and Electronics Engineers, "Standard for Binary Floating-Point Arithmetic", IEEE 754-1985, October 1985, http://grouper.ieee.org/groups/754/

[MS-DTYP] Microsoft Corporation, "Windows Data Types", January 2007.

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

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

[MS-LCID] Microsoft Corporation, "Windows Language Code Identifier (LCID) Reference", July 2007.

[RFC793] Postel, J., "Transmission Control Protocol: DARPA Internet Program Protocol Specification", RFC 793, September 1981, http://www.ietf.org/rfc/rfc0793.txt

[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

[RFC2246] Dierks, T. and Allen, C., "The TLS Protocol Version 1.0", RFC 2246, January 1999, http://www.ietf.org/rfc/rfc2246.txt

[SSL3] Netscape, "SSL 3.0 Specification", http://wp.netscape.com/eng/ssl3/

If you have any trouble finding [SSL3], please check here.

1.2.2 Informative References

[MSDN-MARS] Microsoft Corporation, "Multiple Active Result Sets (MARS) in SQL Server 2005", http://msdn2.microsoft.com/en-us/library/ms345109.aspx

[PIPE] Microsoft Corporation, "Named Pipes", http://msdn2.microsoft.com/en-us/library/aa365590.aspx

[VIA] Intel Corporation, "Intel Virtual Interface (VI) Architecture Developer's Guide", September 1998, ftp://download.intel.com/design/servers/vi/intel.pdf

1.3 Protocol Overview (Synopsis)

The Session Multiplex Protocol is an application protocol that facilitates session management by providing a mechanism to create multiple lightweight communication channels (sessions) over a single physical connection. It does so by multiplexing data streams from different sessions on top of a single reliable stream-oriented transport, such as TCP (as specified in [RFC793]), Named Pipes (see [PIPE]), or **Virtual Interface Architecture** (see [VIA]).

The Session Multiplex Protocol is beneficial in situations where database connections from the **client** and **server** are very synchronous. In this context, "synchronous" implies that the client application can only have one outstanding command/transaction per connection. Rather than incur the expense of creating multiple connections to the server, the Session Multiplex Protocol may be used to allow multiple database queries to be executed simultaneously over a single connection.

The Session Multiplex Protocol provides:

1. The ability to interleave data from several different sessions and preserve message boundaries.

2. A sliding window-based flow control mechanism to facilitate fairness among sessions.

Note that the Session Multiplex Protocol relies upon the underlying transport mechanism to ensure byte-alignment, loss detection and recovery, and reliable delivery. Furthermore, the scheduling algorithm used to enforce fairness between the sessions is an implementation issue for the application implementing the Session Multiplex Protocol.

The following diagram depicts a typical flow of communication in the Session Multiplex Protocol for an arbitrary session.

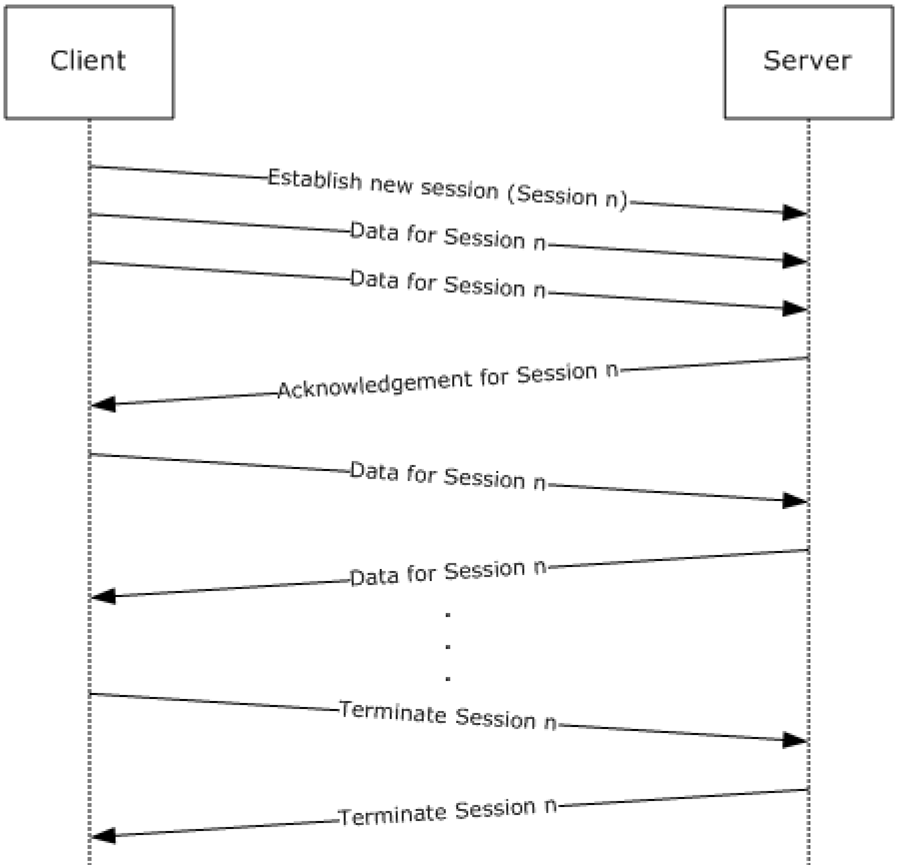

Figure 1: Example of a communication flow in the Session Multiplex Protocol

A maximum sliding window size of 4 packets is used in the Session Multiplex Protocol. The size of the client or server window is conveyed to the peer in the header of every packet sent. In the case where consecutive data packets are sent prior to a response being received, the receiver may send back an acknowledgement packet to the sender to update the window size.

1.4 Relationship to Other Protocols

The Session Multiplex Protocol depends upon an underlying reliable stream-oriented network transport. Optionally, TLS/SSL ([RFC2246] and [SSL3]) may be **inserted** between the Session Multiplex Protocol and transport layer to provide data protection. The **TDS** protocol depends upon the Session Multiplex Protocol when the **MARS** feature is specified. This relationship is depicted in the following diagram.

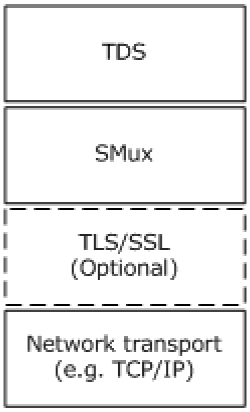

Figure 2: Protocol relationship

1.5 Prerequisites/Preconditions

Throughout this document, it is assumed that the client has already discovered the server and established a network transport connection.

If channel encryption is to be used, then TLS/SSL support must be present on both the client and server machines and a certificate suitable for encryption must be deployed on the server machine.

1.6 Applicability Statement

The Session Multiplex Protocol is appropriate for use to facilitate multiplexing several sessions over a single reliable, physical connection where network or local connectivity is available.

1.7 Versioning and Capability Negotiation

No other version of the Session Multiplex Protocol exists outside of the protocol described in this document. Furthermore, this protocol does not support any capability negotiation.

1.8 Vendor-Extensible Fields

There are no vendor-extensible fields.

1.9 Standards Assignments

There are no standards assignments for the Session Multiplex Protocol.

2 Messages

All integer fields are represented in **little-endian**.

2.1 Transport

The Session Multiplex Protocol is a simple protocol layered above existing reliable transport mechanisms (for example TCP, named pipes) that can be used to create multiple sessions over a single connection. This protocol is defined as a transport-independent mechanism.

2.2 Message Syntax

All Session Multiplex Protocol packets consist of a 16-byte header followed by an optional data payload depending on the packet type.

2.2.1 Header

The 16-byte Session Multiplex Protocol header has the following format:

<table>
<tr><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>1
0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>2
0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>3
0</td><td>1</td></tr>
<tr><td colspan="8">SMID</td><td colspan="8">FLAGS</td><td colspan="16">SID</td></tr>
<tr><td colspan="32">LENGTH</td></tr>
<tr><td colspan="32">SEQNUM</td></tr>
<tr><td colspan="32">WNDW</td></tr>
</table>

SMID (1 byte): This unsigned integer is the Session Multiplex Protocol packet identifier and is always assigned the value 0x53. This indicates that the packet is a Session Multiplex Protocol packet and helps to distinguish it from other protocol packets.

FLAGS (1 byte): This unsigned integer value contains the control flags, as defined in Control Flags, section 2.2.1.1.

SID (2 bytes): This unsigned integer is the session identifier. This is a unique identifier for each session multiplexed over this connection.

LENGTH (4 bytes): This unsigned integer is the length of the Session Multiplex Protocol packet in bytes, including the header.

SEQNUM (4 bytes): This unsigned integer is the Session Multiplex Protocol sequence number for this packet in the session. The first packet on each session has a SEQNUM value of 0x00000000. This then monotonically increases for every DATA packet, up to 0xffffffff, then wraps back to 0x00000000. Sequence numbers are only incremented for DATA packets. For SYN, ACK, and FIN packet types, the sequence number remains stable. If the sequence number deviates from the expected value or if any packet other than a DATA packet increments the SEQNUM value, then an error results and the session closes.

WNDW (4 bytes): This unsigned integer indicates the limit for receive packets. A packet with a sequence number bigger than this will cause the session to close. The difference between this number and the last packet sent is the available window size. The window size MUST be less than or equal to 4.

2.2.1.1 Control Flags

The control flag is one byte after the **SMID** field and it indicates the type of the packet. Only DATA (section 2.2.5) packets have payload data. The meaning of each flag is described below:

0	1	2	3	4	5	6	7
-	-	-	-	DATA	FIN	ACK	SYN

The client MUST NOT send a combination of flags in the same packet. For example, a **FLAGS** field value of 0x06 (ACK + FIN) is an invalid value.

Value	Meaning
SYN 0x01	Indicates that a new connection is to be established (see SYN (section 2.2.2) packet). The session ID for the session will be the number stored in the **SID** field.
ACK 0x02	Used to inform the peer about a change in window size when consecutive, unanswered DATA (section 2.2.5) packets are received. See ACK packet.
FIN 0x04	Indicates that the sending entity will no longer use the session to send data. This essentially closes the pipe in one direction; after the other side also sends a FIN (section 2.2.4) packet of its own, the session is completely closed and the session ID becomes available for new sessions to be allocated.
DATA 0x08	The packet carries user data, after the header (see DATA (section 2.2.5) packet). The length of data buffer is the total Session Multiplex Protocol packet length minus the Session Multiplex Protocol header. For example a **LENGTH** value of 0x25 means 0x15 bytes of user data.

2.2.2 SYN Packet

The SYN packet is sent to indicate that a new connection is to be established. The session ID for the session will be the number stored in the **SID** field. If the SID of an active session is used, then an error will result and the session will close.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
SMID								FLAGS								SID															
LENGTH																															
SEQNUM																															
WNDW																															

SMID (1 byte): This unsigned integer is the Session Multiplex Protocol packet identifier and is always assigned the value 0x53. This indicates that the packet is a Session Multiplex Protocol packet and helps to distinguish it from other protocol packets.

FLAGS (1 byte): This is an unsigned integer containing control flags identifying this as a SYN packet. This field MUST be 0x01.

SID (2 bytes): This unsigned integer contains a session identifier. This is a unique identifier for each session multiplexed over this connection. All subsequent packets in this session MUST use this session identifier.

LENGTH (4 bytes): This unsigned integer contains the length of the SYN Session Multiplex Protocol packet in bytes, including the header. This field MUST be 0x00000010.

SEQNUM (4 bytes): This unsigned integer is the Session Multiplex Protocol sequence number for this packet in the session. The first packet on each session has a SEQNUM of 0x00000000. This then monotonically increases for every DATA packet, up to 0xffffffff, then wraps back to 0x00000000. Sequence numbers are only incremented for DATA packets. For SYN, ACK, and FIN packet types, the sequence number remains stable. If the sequence number deviates from the expected value or if any packet other than a DATA packet increments the SEQNUM, then an error will result and the session will close.

WNDW (4 bytes): This unsigned integer indicates the limit for receive packets. A packet with a sequence number bigger than this will cause the connection to fail. The difference between this number and the last packet sent is the available window size. The window size MUST be less than or equal to 4.

2.2.3 ACK Packet

The ACK packet is used to update the peer with a change in the window size when several consecutive unanswered DATA packets are received. For example, if the client has five packets to send to the server for a single request, then after four packets, it must wait until it receives an updated **WNDW** value before it can transmit further packets; after the server has processed at least one of the packets, it will send a ACK packet to the client with an updated **WNDW** value so that the client sends the last packet and completes the request. If the window size is set to greater than 4 then the session will close.

<table>
<tr><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>1
0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>2
0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>3
0</td><td>1</td></tr>
<tr><td colspan="8">SMID</td><td colspan="8">FLAGS</td><td colspan="16">SID</td></tr>
<tr><td colspan="32">LENGTH</td></tr>
<tr><td colspan="32">SEQNUM</td></tr>
<tr><td colspan="32">WNDW</td></tr>
</table>

SMID (1 byte): This unsigned integer is the Session Multiplex Protocol packet identifier and is always assigned the value 0x53. This indicates that the packet is a Session Multiplex Protocol packet and helps to distinguish it from other protocol packets.

FLAGS (1 byte): This is an unsigned integer containing control flags identifying this as a ACK packet. This field MUST be 0x02.

SID (2 bytes): This unsigned integer contains a session identifier. This is a unique identifier for each session multiplexed over this connection. This MUST be the value that was set when the session was opened.

LENGTH (4 bytes): This unsigned integer specifies the length of the ACKSession Multiplex Protocol packet in bytes, including the header. This field MUST be 0x00000010.

SEQNUM (4 bytes): This unsigned integer is the Session Multiplex Protocol sequence number for this packet in the session. The first packet on each session has a **SEQNUM** value of 0x00000000. This then monotonically increases for every DATA packet, up to 0xffffffff, then wraps back to 0x00000000. Sequence numbers are only incremented for DATA packets. For SYN, ACK, and FIN packet types, the sequence number remains stable. If the sequence number deviates from the expected value or if any packet other than a DATA packet increments the **SEQNUM** value, then an error will result and the session will close.

WNDW (4 bytes): This unsigned integer indicates the limit for receive packets. A packet with a sequence number bigger than this will cause the connection to fail. The difference between this number and the last packet sent is the available window size. The window size MUST be less than or equal to 4.

2.2.4 FIN Packet

The FIN packet is sent to indicate that the sending entity will no longer use the session to send data. This essentially closes the pipe in one direction and after the other side also sends a FIN of its own, the session is completely closed and the session ID becomes available for new sessions to be allocated.

<table>
<tr><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>1
0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>2
0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>3
0</td><td>1</td></tr>
<tr><td colspan="8">SMID</td><td colspan="8">FLAGS</td><td colspan="16">SID</td></tr>
<tr><td colspan="32">LENGTH</td></tr>
<tr><td colspan="32">SEQNUM</td></tr>
<tr><td colspan="32">WNDW</td></tr>
</table>

SMID (1 byte): This unsigned integer is the Session Multiplex Protocol packet identifier and is always assigned the value 0x53. This indicates that the packet is a Session Multiplex Protocol packet and helps to distinguish it from other protocol packets.

FLAGS (1 byte): This is an unsigned integer containing control flags identifying this as a FIN packet. This field MUST be 0x04.

SID (2 bytes): This unsigned integer contains a session identifier. This is a unique identifier for each session multiplexed over this connection. This MUST be the value that was set when the session was opened.

LENGTH (4 bytes): This unsigned integer specifies the length of the FINSession Multiplex Protocol packet in bytes, including the header. This field MUST be 0x00000010.

SEQNUM (4 bytes): This unsigned integer is the Session Multiplex Protocol sequence number for this packet in the session. The first packet on each session has a **SEQNUM** of 0x00000000. This then monotonically increases for every DATA packet, up to 0xffffffff, then wraps back to 0x00000000. Sequence numbers are only incremented for DATA packets. For SYN, ACK, and FIN packet types, the sequence number remains stable. If the sequence number deviates from the expected value or if any packet other than a DATA packet increments the **SEQNUM**, then an error will result and the session will close.

WNDW (4 bytes): This unsigned integer indicates the limit for receive packets. A packet with a sequence number bigger than this will cause the connection to fail. The difference between this number and the last packet sent is the available window size. The window size MUST be less than or equal to 4.

2.2.5 DATA Packet

The DATA packet carries data after the header. The length of the data is the total Session Multiplex Protocol packet length minus the Session Multiplex Protocol header length. For example a **LENGTH** value of 0x25 means 0x15 bytes of user data.

<table>
<tr><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>10</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>20</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>30</td><td>1</td></tr>
<tr><td colspan="8">SMID</td><td colspan="8">FLAGS</td><td colspan="16">SID</td></tr>
<tr><td colspan="32">LENGTH</td></tr>
<tr><td colspan="32">SEQNUM</td></tr>
<tr><td colspan="32">WNDW</td></tr>
<tr><td colspan="32">DATA (variable)</td></tr>
<tr><td colspan="32">...</td></tr>
</table>

SMID (1 byte): This unsigned integer is the Session Multiplex Protocol packet identifier and is always assigned the value 0x53. This indicates that the packet is a Session Multiplex Protocol packet and helps to distinguish it from other protocol packets.

FLAGS (1 byte): This is an unsigned integer containing control flags identifying this as a DATA packet. This field MUST be 0x08.

SID (2 bytes): This unsigned integer contains a session identifier. This is a unique identifier for each session multiplexed over this connection. This MUST be the value that was set when the session was opened.

LENGTH (4 bytes): This unsigned integer specifies the length of the DATA Session Multiplex Protocol packet in bytes, including the header. This field MUST be greater than 0x00000010.

SEQNUM (4 bytes): This unsigned integer is the Session Multiplex Protocol sequence number for this packet in the session. The first packet on each session has a **SEQNUM** value of 0x00000000. This then monotonically increases for every DATA packet, up to 0xffffffff, then wraps back to 0x00000000. Sequence numbers are only incremented for DATA packets. For SYN, ACK, and FIN packet types, the sequence number remains stable. If the sequence number deviates from the expected value or if any packet other than a DATA packet increments the **SEQNUM** value, then an error will result and the session will close.

WNDW (4 bytes): This unsigned integer indicates the limit for receive packets. A packet with a sequence number bigger than this will cause the connection to fail. The difference between this number and the last packet sent is the available window size. The window size MUST be less than or equal to 4.

DATA (variable): This is the data packet. It is (**LENGTH** - 16) bytes long.

3 Protocol Details

This section describes the important elements of the client software and the server software necessary to support the Session Multiplex Protocol.

The Session Multiplex Protocol is a largely symmetric protocol and obeys the same rules and semantics on both the client and the server. As a result, the description for both the client and server roles are contained in the Common Details section with section 3.1.4.3 applying only to the client and section 3.1.5.1.1 applying only to the server.

3.1 Common Details

The Session Multiplex Protocol MUST be layered on top of a reliable, in-order, connection-oriented transport layer such as TCP (as specified in [RFC793]), Named Pipes (see [PIPE]), or Virtual Interface Architecture (see [VIA]). The timing of Session Multiplex Protocol initiation on both ends of a transport connection must be negotiated through other protocols, before it starts to operate. The shutdown sequence can be triggered by either the higher layer or by fatal events internal to the Session Multiplex Protocol. The peer is notified of the shutdown by the closing of the transport connection.

The Session Multiplex Protocol has the notion of a client and a server during session establishment, in which a session MUST be initiated by a client; see section 3.1.4.1. After a session is established, both endpoints of a session can be used by the higher layer to send and receive data symmetrically. Either the client or the server can initiate connection termination by sending a FIN packet. Upon receiving a FIN packet, an endpoint sends a FIN packet back to its peer if a FIN packet has not yet been sent, and then recycles its session object, see sections 3.1.4.3 and 3.1.5.1.3. Session endpoints MUST obey the flow control algorithm in section 3.1.1.3 when sending and receiving DATA packets.

A session progresses through a series of states during its lifetime. The states are: LISTENING (server only), SESSION ESTABLISHED, FIN SENT, FIN RECVED and CLOSED. CLOSED is the state in which a session does not exist.

A session progresses from one state to another in response to the following events:

- User calls: Open (client only), Send, Receive, Close, Initialize, and Shutdown
- Local events: transport_disconnect and system_failure
- Incoming packets: SYN (client only), DATA, ACK and FIN

The following state diagrams illustrate the progress of the lifetime of a session on the client and the server. The diagrams are only summaries and SHOULD NOT be taken as the total specification. They do not include error events and state changes within an established state. Note that the client and server diagrams are very similar, with the notable exception of the difference in reaching the "Session Established" state.

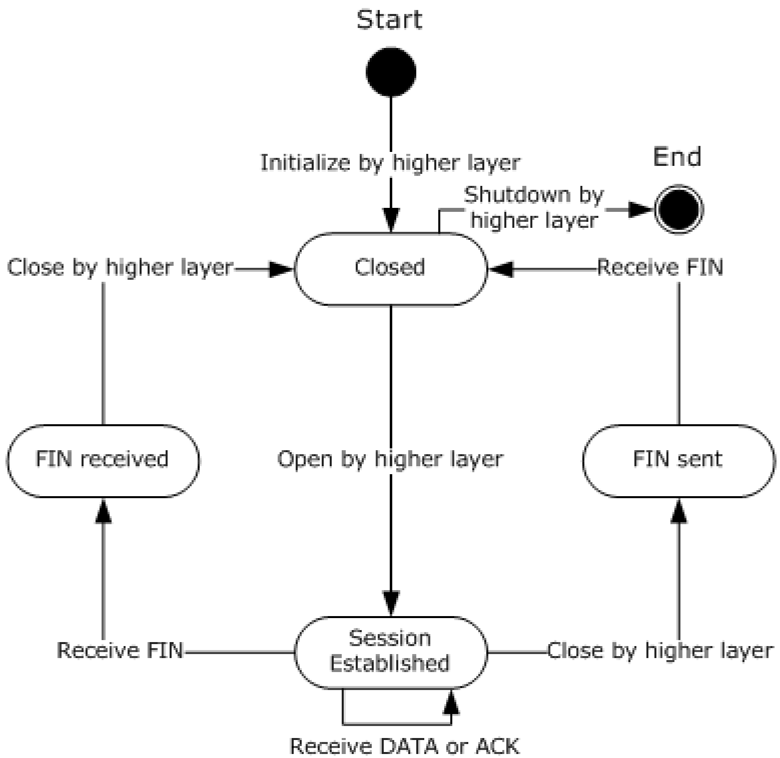

Figure 3: Session Multiplex Protocol client state machine

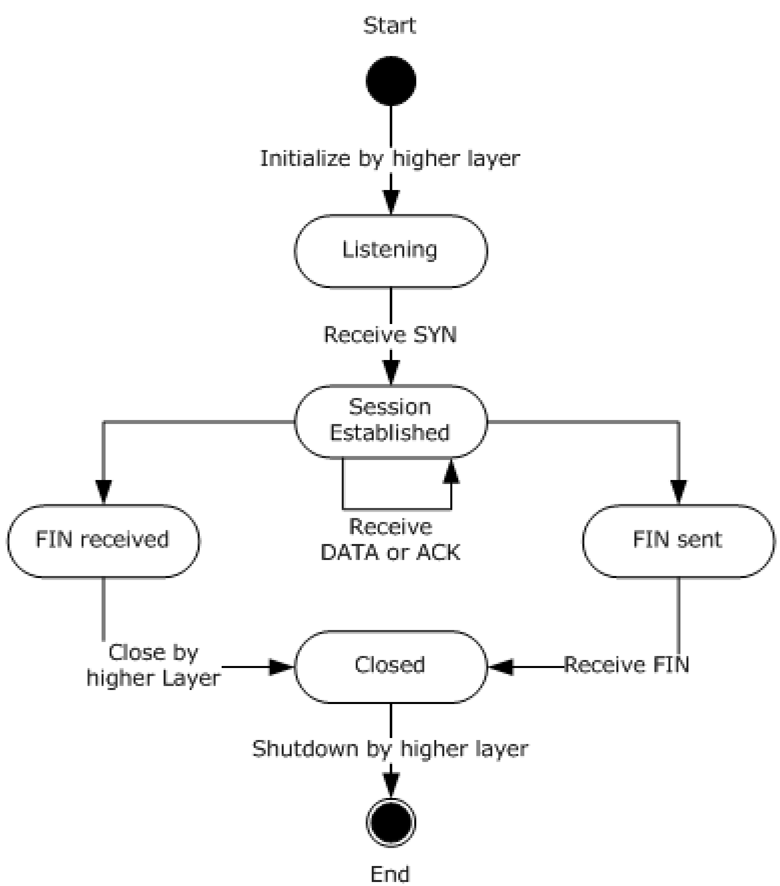

Figure 4: Session Multiplex Protocol server state machine

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 Per Transport Connection Structure

The following are structures that are required per transport connection. They are needed to ensure the uniqueness of active Session Multiplex Protocol sessions.

SMUX.SessionTable: A table of connected sessions that have been established on this transport connection. The table MUST be uniquely indexed by Session.SessionId.

SMUX.Connection: A reference to the physical connection on which the session is established.

3.1.1.2 Per Session Structure

The following are structures that are required per session. They are needed to implement the flow control algorithm and for connection management.

Session.SessionId: A numeric value which uniquely identifies the session within the scope of the transport connection over which the session was established.

Session.Smux: A reference to the per transport connection structure on which the session is created.

Session.SeqNumForSend: A 32-bit unsigned integer that monotonically increases for every session DATA packet to send.

Session.HighWaterforSend: A 32-bit unsigned integer that keeps track of the peer's window obtained through the header field **WNDW** of the received session packet. Any packet to send MUST NOT contain a **SEQNUM** value bigger than HighWaterforSend.

Session.SeqNumforRecv: A 32-bit unsigned integer that keeps track of the peer's session DATA sequence number obtained from the header field **SEQNUM**. This number is used to compare with the received packet. The **SEQNUM** value of a DATA packet MUST be equal to the SeqNumforRecv value plus 1. The **SEQNUM** value of a packet other than a DATA packet MUST be equal to SeqNumforRecv.

Session.HighWaterforRecv: A 32-bit unsigned integer that tracks the receiver's high-water mark of the receiver buffer window. It is used to assign the **WNDW** field of each sent packet.

Session.LastHighWaterforRecv: A 32-bit unsigned integer that tracks the **WNDW** field of the last sent packet. It is used to implement a selective ACK algorithm and is optional.

Session.ReceivePacketQueue: A queue that buffers received packets.

Session.FinSent: A Boolean value that indicates that the session endpoint has initiated a session termination sequence and a FIN packet has been sent.

Session.FinRecved: A Boolean value that indicates that the session endpoint has received a FIN packet.

Session.BadConnection: A Boolean value that indicates whether the transport connection is disconnected.

3.1.1.3 Flow Control Algorithm

The Session Multiplex Protocol provides a means for the receiver to govern the amount of data sent by the sender. This is achieved by returning a "window" with every ACK or DATA packet on reverse direction indicating a range of acceptable sequence numbers beyond the last DATA packet successfully received. The window indicates an allowed number of DATA packets that the sender may transmit before receiving further permission.

Variables used in flow control include sender variables (Session.SeqNumforSend and Session.HighWaterforSend) and receiver variables (Session.SeqNumforRecv, Session.HighWaterforRecv and LastHighWaterforRecv). The following sections show the relationships of these variables in the sequence number space. The sequence number is a 32-bit unsigned integer that is allowed to wrap.

3.1.1.3.1 Sender

The following are relationships between session variables for the sender.

```
---------------|------------------------|--------------> +
```

SeqNumforSend HighWaterforSend

1. The following condition MUST be true: SeqNumForSend is less than or equal to HighWaterForSend.
2. The DATA packet MUST NOT be sent if SeqNumforSend is equivalent to HighWaterforSend.
3. Otherwise, the DATA packet is sent and then SeqNumforSend is incremented by 1.
4. On receiving a packet, HighWaterForSend equals the received packet.WNDW value.

Note that window size equals HighWaterforSend minus SeqNumforSend; the window is said to be closed when the window size is zero. The maximum window size for the current implementation is 4.

3.1.1.3.2 Receiver

The following are relationships between session variables for the receiver.

---------------|------------------------|---------------> +

SeqNumforRecv HighWaterforRecv

1. The following condition MUST be true:. SeqNumforRecv is less than or equal to HighWaterforRecv.
2. When the higher layer retrieves a packet from a session endpoint, increment HighWaterforRecv by 1.
3. When sending a packet, the packet.WNDW value equals HighWaterforRecv.
4. When receiving a packet, the SeqNumforRecv value equals packet.SEQNUM.

Note that the window size equals HighWaterforRecvless the value of SeqNumforRecv;

3.1.1.3.3 Update Sender's HighWaterforSend Variable Using a ACK Packet

The sender's HighWaterforSend variable is updated by either a DATA packet or ACK packet on reverse direction. The simple choice is to send a ACK for every packet retrieved by a higher layer. An algorithm that can delay ACK, the so-called selective ACK algorithm, is described as the following.

---------------|-------------------|--------------------> +

LastHighWaterforRecv HighWaterforRecv

1. The following condition MUST be true: LastHighWaterforRecv is less than HighWaterforRecv.
2. When sending a packet, the LastHighWaterforRecv variable equals the sent Packet.WNDW value.
3. When a DATA packet is retrieved by a higher layer, HighWaterforRecv equals HighWaterforRecv plus 1.
4. A ACK packet can be sent for each DATA packet retrieved by the higher layer; alternatively, in a simple selective acknowledgement algorithm, a ACK packet can be sent after every other DATA packet retrieved by the higher layer as "if (HighWaterforRecv minus LastHighWaterforRecv is greater than or equal to X) send ACK", and thereby reduce the cost, where the X MUST be smaller than the maximum window size, which is 4 for the current implementation, and X may be chosen as 2.

3.1.2 Timers

There are no timers in the Session Multiplex Protocol. It assumes a reliable transport and the eventual delivery of messages. In the event of an error from the transport connection, the Session Multiplex Protocol recycles all session objects associated with the failed transport connection. Idle sessions are kept open until the higher layer closes them or an error in transport connection happens.

3.1.3 Initialization

3.1.3.1 Per Transport Connection Structure

A per transport connection structure MUST be initialized on both client and server endpoints of the transport connection before the Session Multiplex Protocol can operate. The exact timing MUST be negotiated by the other protocol.

3.1.3.2 Per Session Structure

A per session structure MUST be initialized as in the following table.

Variable	Value
Session.SeqNumforSend	0
Session.HighWaterforSend	4
Session.SeqNumforRecv	0
HighWaterforRecv	4
Session.FinSent	False
Session.FinRecved	False
Session.BadConnection	False

If the selective acknowledgement algorithm is used, LastHighWaterforRecv equals 4.

3.1.4 Higher-Layer Triggered Events

3.1.4.1 Initiating the Session Multiplex Protocol on a Transport Endpoint

Both client and server MUST initiate the Session Multiplex Protocol on both ends of a transport connection before it can operate. It initiates a per transport connection structure as specified in section 3.1.3.1. After initiation, the client enters a CLOSED state and the server enters a LISTENING state.

3.1.4.2 Shutting Down the Session Multiplex Protocol on a Transport Endpoint

Both client and server can shut down the Session Multiplex Protocol. For either server or client, the following sequence of actions MUST happen during shutdown.

1. MUST walk through all session objects in SMUX.SessionTable and mark them as bad connections (that is, Session.BadConnection equals True) so that any call to access a session object from a

higher layer (that is, send and receive) will result in failure and the higher layer MUST terminate (close call) the session thereafter.

2. MUST close the transport connection so that the peer gets notified of the shutdown event.
3. The higher layer terminates a session by a Close call, and the session objects are removed from SessionTable and freed.
4. The higher layer calls shutdown if the shutdown is triggered by internal events, as in section 3.1.7.
5. The SessionTable is freed after all session objects are removed and freed.

Note that no Session Multiplex Protocol packet is sent to the peer during the shutdown sequence. The peer SHOULD be notified by transport_disconnect event if it happens.

3.1.4.3 Initiating Session Establishment (Open Call)

The session establishment sequence MUST be initiated by the client. It MUST be initiated after the Session Multiplex Protocol per transport connection structure is initialized.

A session establishment event triggers the following sequence of actions:

1. Choose a unique session ID per transport connection.
2. Create a session object and insert it into the session table.
3. Send a SYN packet to the server.

3.1.4.4 Sending Messages to Peer and Receiving Messages from Peer (Send and Receive Call)

If a message is being sent, the higher layer MUST provide a buffer that contains the message.

If a message cannot be sent to its peer because the receiver window closes, such as when Session.SeqNumforSend is equivalent to Session.HighWaterforSend, the session endpoint of the sender may choose to do one of the following:

- Buffer the message in a local buffer and send it at a later time.
- Block the higher layer until a message is sent.
- Return the error (for example, ERROR_WOULDBLOCK) to the higher layer so that the higher layer may choose to send messages at a later time.

Otherwise, a DATA packet is sent.

The higher layer can retrieve messages from a session endpoint. Data received from the peer but not yet retrieved by the higher layer is buffered by the session endpoint of the receiver.

If data has not yet arrived from its peer, the retrieve attempt by the higher layer may:

- Fail (for example, with ERROR_IO_PENDING).
- Block until a message arrives.

The successful retrieval of a message triggers HighWaterforRecv to be incremented by 1 and MAY trigger an ACK packet be sent to the peer according the selective acknowledgement algorithm in section 3.1.1.3.3.

3.1.4.5 Session Termination (Close Call)

When a session termination is triggered by a higher layer, the following sequence of actions MUST happen:

1. If Session.FinRecved is equivalent to False, set Session.FinSent to True and send the FIN packet.

2. If Session.FinRecved is equivalent to True, send the FIN packet, remove the session object from the session table, and recycle the session object.

Note that the session object cannot be recycled until the client or server receives a FIN packet from its peer, as described in section 3.1.5.1.3. Also, it is important for the client to both receive a FIN packet and send a FIN packet (the order doesn't matter) before removing the session object from the session table. This prevents a new session from reusing the session ID before the server has cleaned up the session object.

3.1.5 Message Processing Events and Sequencing Rules

3.1.5.1 Receiving a Packet

The following MUST happen when receiving a packet.

1. Parse out the packet header.

2. Locate a session object in the SessionTable according to the **packet.SID**.

3. If the session object doesn't exist and the packet is NOT a SYN packet, raise a fatal error to the higher layer and close the transport connection. Otherwise, process the SYN packet.

4. If the session object is located, the following conditions MUST be met; otherwise, raise a fatal exception and close the transport connection.

   - **packet.type** does not equal SYN
   - **packet.WNDW** is greater than or equal to Session.HighWaterforSend
   - **packet.SID** is equivalent to Session.SessionID
   - **packet.SEQNUM** is less than or equal to Session.HighWaterforRecv

5. Type equals **packet.Type**.

6. If Type equals DATA, parse out the entire data packet according to the packet.LENGTH value.

In receiving SYN, DATA, ACK, or FIN packets, the following MUST happen.

3.1.5.1.1 Receiving a SYN packet

This logic applies to the server only.

1. Create a session object with session ID equal to the **SID** contained in the SYN packet.

2. Insert the session object in the SMUX.SessionTable.

3.1.5.1.2 Receiving a DATA packet

When a DATA packet is received:

1. If a higher layer posted a receive, finish that receive with the data in the packet; otherwise, buffer the packet in the Session.ReceivePacketQueue variable.

2. If the **packet.WNDW** value is greater than Session.HighWaterforSend, either the client or the server SHOULD send the pending write if any and MUST update Session.HighWaterforSend to the **packet.WNDW** value.

3.1.5.1.3 Receiving a ACK packet

When a ACK packet is received:

- If the **packet.WNDW** value is greater than Session.HighWaterforSend, either the server or the client SHOULD send the pending write if any and MUST update Session.HighWaterforSend to the **packet.WNDW** value.

3.1.5.1.4 Receiving a FIN packet

When a FIN packet is received:

1. If Session.FinSent is equivalent to FALSE, Session.FinRecved equals TRUE;

2. If Session.FinSent is equivalent to TRUE, remove the session object from SMUX.SessionTable and recycle the session object.

3.1.6 Timer Events

There is no timer in the Session Multiplex Protocol.

3.1.7 Other Local Events

In case of the following fatal events, the Session Multiplex Protocol resets itself with the sequence of actions as in section 3.1.4.2.

The fatal events include:

1. Transport disconnects.

2. Packet parse error if the packet received doesn't obey the rules.

3. System failures, such as failure of memory allocation.

4 Protocol Examples

This section gives a few examples of Session Multiplex Protocol packets, depending on the operation being performed.

4.1 New Session

This example shows a new session.

<table>
<tr><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>1
0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>2
0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>3
0</td><td>1</td></tr>
<tr><td colspan="8">SMID</td><td colspan="8">FLAGS</td><td colspan="16">SID</td></tr>
<tr><td colspan="32">LENGTH</td></tr>
<tr><td colspan="32">SEQNUM</td></tr>
<tr><td colspan="32">WNDW</td></tr>
</table>

SMID (1 byte): 0x53

FLAGS (1 byte): 0x01 (SYN)

SID (2 bytes): 0x0000 (First Session Multiplex Protocol session on this connection)

LENGTH (4 bytes): 0x00000010 (SYN packet doesn't have any payload)

SEQNUM (4 bytes): 0x00000000 (Initial packet for this session)

WNDW (4 bytes): 0x00000004 (Default of 4 receive buffers posted)

4.2 Update Window - ACK

This example shows update window.

<table>
<tr><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>1
0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>2
0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>3
0</td><td>1</td></tr>
<tr><td colspan="8">SMID</td><td colspan="8">FLAGS</td><td colspan="16">SID</td></tr>
<tr><td colspan="32">LENGTH</td></tr>
<tr><td colspan="32">SEQNUM</td></tr>
<tr><td colspan="32">WNDW</td></tr>
</table>

SMID (1 byte): 0x53

FLAGS (1 byte): 0x02 (ACK)

SID (2 bytes): 0x0005 (Session ID 5)

LENGTH (4 bytes): 0x00000010 (ACK packet doesn't have a payload)

SEQNUM (4 bytes): 0x00000010

WNDW (4 bytes): 0x00000012

4.3 First Command on a Session

This example shows first command in a session.

<table>
<tr><th>0</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>10</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>20</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>30</th><th>1</th></tr>
<tr><td colspan="8">SMID</td><td colspan="8">FLAGS</td><td colspan="16">SID</td></tr>
<tr><td colspan="32">LENGTH</td></tr>
<tr><td colspan="32">SEQNUM</td></tr>
<tr><td colspan="32">WNDW</td></tr>
<tr><td colspan="32">DATA (variable)</td></tr>
<tr><td colspan="32">...</td></tr>
</table>

SMID (1 byte): 0x53

FLAGS (1 byte): 0x08 (DATA)

SID (2 bytes): 0x0005 (Session ID 5)

LENGTH (4 bytes): 0x00000060

SEQNUM (4 bytes): 0x0000001

WNDW (4 bytes): 0x0000004

DATA (variable): 0x01 01 00 50 00 00 01 00 16 00 00 00 12 00 00 00 02 00 00 00 00 00 00 00 00 00 01 00 00 00 53 00 45 00 54 00 20 00 51 00 55 00 4F 00 54 00 45 00 44 00 5F 00 49 00 44 00 45 00 4E 00 54 00 49 00 46 00 49 00 45 00 52 00 20 00 4F 00 46 00 46 00 (TDS request)

4.4 Last Command on a Session

This example shows last command in a session.

<table>
<tr><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>1
0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>2
0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>3
0</td><td>1</td></tr>
<tr><td colspan="8">SMID</td><td colspan="8">FLAGS</td><td colspan="16">SID</td></tr>
<tr><td colspan="32">LENGTH</td></tr>
<tr><td colspan="32">SEQNUM</td></tr>
<tr><td colspan="32">WNDW</td></tr>
</table>

SMID (1 byte): 0x53

FLAGS (1 byte): 0x04 (FIN)

SID (2 bytes): 0x0005 (Session ID 5)

LENGTH (4 bytes): 0x00000010 (FIN packet doesn't have a payload)

SEQNUM (4 bytes): 0x0000023

WNDW (4 bytes): 0x0000013

5 Security

5.1 Security Considerations for Implementers

There are no special security considerations for this protocol.

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 Server 2003
- Windows Vista
- Windows XP
- Windows 2000
- Windows NT

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.

7 Index

A

Abstract data model
ACK packet
Applicability

C

Capability negotiation
Control_Flags packet

D

Data model - abstract
DATA packet

F

Fields - vendor-extensible
FIN packet
First_Command_on_a_Session packet

G

Glossary

H

Header packet

I

Informative references
Introduction

L

Last_Command_on_a_Session packet

M

Messages
  overview
  syntax
  transport

N

New_Session packet
Normative references

O

Overview (synopsis)

P

Preconditions
Prerequisites

R

References
  informative
  normative
  overview
Relationship to other protocols

S

SYN packet
Syntax - message

T

Transport - message

U

Update_Window-ACK packet

V

Vendor-extensible fields
Versioning

W

Windows behavior