[MC-MQSRM]:
Message Queuing (MSMQ):
SOAP Reliable Messaging Protocol (SRMP)

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.

Date	Revision History	Revision Class	Comments
01/25/2008	1.0	Major	New sections.

Table of Contents

1 Introduction

This document specifies the Message Queuing (MSMQ): SOAP Reliable Messaging Protocol (SRMP), which defines a mechanism for reliably transferring **messages** between two **message queues** located on two different hosts. The document also specifies how **MSMQ** uses Pragmatic General Multicast (PGM) to reliably multicast SRMP messages between a sending message queue and a set of receiving message queues. SRMP uses SOAP 1.1 over HTTP, as specified in [SOAP1.1], to transport data, but augments it with additional levels of acknowledgment that ensure that the messages are reliably transferred irrespective of connection, application, or node failures. For more information about MSMQ architecture and concepts, see [MS-MQMA] and [MS-MQMQ].

Familiarity with Internet messaging standards, such as HTTP, MIME, **XML**, and SOAP, is required for a complete understanding of this specification. Also, familiarity with the basic concepts of MSMQ is required.

This protocol is implemented by all Windows releases from Windows XP onward.

1.1 Glossary

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

Coordinated Universal Time (UTC)
Globally Unique Identifier (GUID)
Universal Unique Identifier (UUID)
UTF-8
XML

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

Connector Queue
Dead-Letter Queue
Message
Message Body
Message Packet Header
Message Queue
Microsoft Message Queuing (MSMQ)
Outgoing Queue
Private Queue
Public Queue
Queue
Queue Journal
Queue Manager (QM)
System Queue
Transactional Message
Transactional Queue

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

Sequence
Source Journaling

The following terms are specific to this document:

Durable Message: A **message** that is written to a stable store during processing to ensure persistence during computer failure or restart.

Regular Message: A **message** that is stored only in computer memory while being processed and that will not survive a computer failure or restart.

Stream: A **sequence** of **messages** whose delivery is guaranteed exactly once and in order.

Stream Message: A **durable message** that is delivered to the receiver exactly once and in **sequence** with other **messages** sent on the **stream**.

Stream Receipt: An acknowledgment **message** that indicates the in-order receipt of **messages** that make up a **stream**.

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.

[ISO-8601] International Organization for Standardization, "Data Elements and Interchange Formats - Information Interchange - Representation of Dates and Times", ISO 8601:2004, December 2004, http://www.iso.org/iso/en/CatalogueDetailPage.CatalogueDetail?CSNUMBER=40874&ICS1=1&ICS2=140&ICS3=30

Note There is a charge to download the specification.

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

[MS-MQMA] Microsoft Corporation, "Message Queuing (MSMQ): Architecture Protocol Specification", August 2007.

[MS-MQMQ] Microsoft Corporation, "Message Queuing (MSMQ): Data Structures", August 2007.

[MS-MQQB] Microsoft Corporation, "Message Queuing (MSMQ): Message Queuing Binary Protocol Specification", August 2007.

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

[RFC2045] Freed, N., et al., "Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies", RFC 2045, November 1996, http://ietf.org/rfc/rfc2045.txt

[RFC2046] Freed, N. and Borenstein, N., "Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types", RFC 2046, November 1996, http://ietf.org/rfc/rfc2046.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

[RFC2387] Levinson, E., "The MIME Multipart/Related Content-type", RFC 2387, August 1998, http://ietf.org/rfc/rfc2387.txt

[RFC2616] Fielding, R., et al., "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2616, June 1999, http://www.ietf.org/rfc/rfc2616.txt

[RFC2818] Rescorla, E., "HTTP Over TLS", RFC 2818, May 2000, http://www.ietf.org/rfc/rfc2818.txt

[RFC3208] Speakman, T., Crowcroft, J., Gemmell, J., Farinacci, D., Lin, S., Leshchiner, D., Luby, M., Montgomery, T., Rizzo, L., Tweedly, A., Bhaskar, N., Edmonstone, R., Sumanasekera, R., and Vicisano, L., "PGM Reliable Transport Protocol Specification", RFC 3208, December 2001, http://www.ietf.org/rfc/rfc3208.txt

[RFC3548] Josefsson, S., Ed., "The Base16, Base32, and Base64 Data Encodings", RFC 3548, July 2003, http://www.ietf.org/rfc/rfc3548.txt

[RFC4122] Leach, P., Mealling, M., and Salz, R., "A Universally Unique Identifier (UUID) URN Namespace", RFC 4122, July 2005, http://www.ietf.org/rfc/rfc4122.txt

[SOAP1.1] Box, D., Ehnebuske, D., Kakivaya, G., Layman, A., Mendelsohn, N., Nielsen, H. F., Thatte, S., and Winer, D., "Simple Object Access Protocol (SOAP) 1.1", May 2000, http://www.w3.org/TR/2000/NOTE-SOAP-20000508/

[W3C-XSD] World Wide Web Consortium, "XML Schema Part 2: Datatypes Second Edition", October 2004, http://www.w3.org/TR/2004/REC-xmlschema-2-20041028/

[XML1.0] Bray, T., Paoli, J., Sperberg-McQueen, C.M., and Maler, E., "Extensible Markup Language (XML) 1.0 (Second Edition)", W3C Recommendation, October 2000, http://www.w3.org/TR/2000/REC-xml-20001006

1.2.2 Informative References

[MSDN-WSROUTING] Microsoft Corporation, "Web Services Routing Protocol (WS-Routing)", Nielsen, H.F., and Thatte, S., 2001, http://msdn2.microsoft.com/en-us/library/ms951249.aspx

1.3 Protocol Overview (Synopsis)

1.3.1 Introduction

The SRMP is used by a client to reliably transfer messages to a server. Specifically, it is used by the MSMQ **Queue Manager (QM)** service on the sender to send a message or a message **stream** to the QM service on the receiver by way of a Web service. The protocol is stateless when exchanging single messages and stateful when sending multiple messages as part of a message stream. The protocol uses SOAP 1.1, as referenced in [SOAP1.1], as its message format. For its transport, the protocol uses HTTP 1.1, as referenced in [RFC2616], or PGM, as referenced in [RFC3208], depending on whether the message is unicast or multicast. SRMP enhances SOAP with additional levels of acknowledgment that ensure that messages are reliably transferred, irrespective of connection, application, or node failures.

1.3.2 Message Queuing

Message queuinq is a communications service that asynchronously and reliably passes messages between client applications running on different hosts. In message queuinq, clients send application messages to a **queue** and/or consume application messages from a queue. The queue provides persistence for messages, which enables them to survive across application restarts, and allows sending and receiving applications to operate asynchronously. Queues are typically hosted by a communications service called a queue manager (QM).

Implementing the QM as a separate service allows client applications to exchange queued messages asynchronously and eliminates the need for the client applications to execute at the same time.

Message queuinq is designed to send messages asynchronously to computers that are temporarily unavailable. When sending a message, the QM indicates to the client application that the sending operation has succeeded as soon as the message is created with valid properties and placed in an **outgoing queue**. The message remains in the outgoing queue until it is delivered to its destination or until the message expires. Note that the sending operation does not immediately deliver the message. It just stores the message in a queue to be delivered asynchronously by the QM.

QMs handle message delivery by continually checking for messages in all the local outgoing queues. When it finds messages, the QM attempts to transmit the messages to their destinations. If the message does not reach its destination queue or is discarded before a receiving application retrieves it, the QM on the sending side does not return any information to the sending application. Applications can obtain information about message delivery from acknowledgment messages that the destination host sends back, as well as from **dead-letter** and **queue journals** for messages sent. For more information on MSMQ architecture, see [MS-MQMA].

1.3.3 SRMP

The SRMP defines a mechanism for reliably transferring messages between QMs located on two different hosts. The protocol does not define the QM or its interface to client applications.

SRMP is designed as a direct alternative to the Message Queuing Binary Protocol between QMs, as specified in [MS-MQQB]. In contrast to the Message Queuing Binary Protocol, which implements message transfer directly between two QMs, SRMP requires a Web service on the receiving computer. In order for the receiver to send acknowledgments back to the sender, SRMP also requires the presence of a Web service on the sending computer. Because HTTP 1.1 is used as a transport protocol, a Web service is required.

Microsoft introduced SRMP in Windows XP (2001) and Windows Server 2003 as part of MSMQ version 3.0.

1.3.4 Message Structure

A typical message exchanged in a message queuinq system includes a **message packet header**, which contains a set of message properties, or metadata, about the message. The message packet header is followed by a distinguished property, called the **message body**, which contains the application payload. In SRMP, the message body is sent as a multipart MIME [RFC2387] attachment, not as part of the SOAP 1.1 message. Thus, the SOAP 1.1 message acts as the message packet header with its body element empty. This allows the sending of multimedia contents in application messages. For more information on MIME, see [RFC2045] and [RFC2046].

1.3.5 User Message Types

Messages sent using the SRMP can be one of three types: **regular**, **durable**, or stream. The type of message, and how it is delivered, depends on whether you want better performance with minimal resources (as with regular messaging) or reliability and recovery after a failure (which durable and streaming messages provide).

1.3.5.1 Regular Messages

Messages sent as regular messages are stored in RAM during transfer and delivery to the destination queue until they are received. This provides fast performance, but the messages are not recoverable if the computer on which the messages reside fails. Notably, this means that regular messages can

be lost when the QM service is stopped. Regular messages are not guaranteed to be delivered only once or in order.

Regular messages can, like durable or **stream messages**, survive a network failure. For example, if the client sends regular messages, and the link between the QM and the target computer fails, the QM continues to store the messages in its memory and retries the connection. However, if the client process fails before the link is restored, the undelivered regular messages are lost. Likewise, regular messages on a server are lost in the event of a process failure.

Regular messages correspond to **express messages**, as described in [MS-MQQB] section 1.3.2.1.1.

1.3.5.2 Durable Messages

Messages sent as durable messages are written to stable storage on both the sending and receiving computer. After delivery to the destination queue, durable messages are stored on disk until a user application accesses them. This makes delivery somewhat slower than with regular messages, but ideal when persistence through service restart or failure is required. If a computer fails or is shut down while sending messages, the messages are stored on disk. When the computer is restarted and the QM service restarts, the sending process automatically resumes. Durable messages are not guaranteed to be delivered only once or in order.

Durable messages correspond to **recoverable messages**, as described in [MS-MQQB] section 1.3.2.1.2.

1.3.5.3 Stream Messages

A stream message is a durable message that has exactly-once-and-in-order (EOIO) delivery guarantees. When delivering stream messages, the SRMP utilizes an additional level of acknowledgment to guarantee that messages arrive only once and in the correct order.

Stream messages are intended to be used in situations where the QM has captured one or more messages under a transaction and subsequently uses SRMP to transfer the messages to a QM on a different host. A QM uses SRMP to transfer messages after a transaction has committed. SRMP does not participate in transaction processing on the client or server.

SRMP does not mandate the implementation details of the transactional capture of messages, as long as the external behavior of a QM is consistent with that specified in this document.

Stream messages correspond to **transactional messages**, as described in [MS-MQQB] section 1.3.2.1.3.

1.3.6 Message Queues

A queue is a logical data structure containing an ordered list of zero or more messages. A QM maintains a set of queues that hold messages. The QM requires a set of predefined or **system queues** (defined as follows) that are referenced throughout this document. A QM configuration defines a set of user queues that are the targets for messages sent using SRMP.

Messages transferred using SRMP are addressed to specific queues by name. SRMP identifies queues using the formats specified in [MS-MQMQ] section 2.1. The MSMQ: SOAP Reliable Messaging Protocol does not mandate the implementation details of queues, as long as their external behaviors are consistent with those described in this document.

A queue can be **transactional** or nontransactional. A transactional queue accepts only stream messages, while a nontransactional queue accepts only regular and durable messages. A

transactional queue requires persistent storage of messages and guaranteed consistency through process or node failure.

1.3.6.1 System Queues

Queues that all QMs must support are referred to as system queues. System queues include the following types of queues:

- Dead-letter queues contain messages that a host sent with a request for negative **source journaling** and could not be delivered. Dead-letter queues can be implemented as transactional or nontransactional.
- Outgoing queues contain messages that must be sent to specific destination addresses. Messages remain in outgoing queues until they can be transferred to their respective destination queues.
- **Connector queues** are temporary locations to store messages that are forwarded to foreign messaging systems. Typically, a connector service running on a server waits for messages to arrive in one or more connector queues and forwards them to the foreign messaging systems. A connector service is application-defined and is not specified by SRMP.
- Queue journals contain copies of the messages sent from hosts when positive source journaling is requested by message queuinq applications.

1.3.7 Source Journaling

Source journaling is the process of storing copies of outgoing messages on a source computer. Source journaling is selected on a per-message basis and is programmatically implemented as a property set by a message queuinq application. Source journaling can be used to track messages that were sent successfully, messages that could not be delivered, or both. By default, Source journaling is not selected.

There are two types of source journaling: positive source journaling and negative source journaling, as described below.

1.3.7.1 Positive Source Journaling

Positive source journaling tracks successfully sent messages by placing message copies in the local host queue journal.

1.3.7.2 Negative Source Journaling

Negative source journaling tracks unsuccessfully sent messages by placing message copies in the local host dead-letter queue. When a message queuinq application requests negative source journaling, messages are processed, depending on the message type.

For nontransactional messages, a copy of the message is placed in the local host dead-letter queue if the source QM on the host cannot transfer the message to the destination host QM.

For transactional messages, a copy of the message is placed in the transactional dead-letter queue of the local host only if message queuinq does not confirm that the message was retrieved from its destination queue.

1.3.8 Internal Receipts

Internal receipts are system-generated protocol messages that the receiving QM at the final destination of a message sends to the sending QM to acknowledge receipt (or other processing) of a

user message. SRMP uses internal receipts to enforce guarantees, such as EOIO delivery. Retransmission of messages may occur when an internal receipt is not received within a specific period of time.

The protocol utilizes the following types of internal receipts:

- **Delivery Receipt**: This message acknowledges receipt of regular and durable user messages. Delivery receipts correspond to **session acknowledgments**, as described in [MS-MQQB] section 1.3.5.1.
- **Commitment Receipt**: This message acknowledges that a delivered message has been removed from the destination queue. Removal from the destination queue could be the result of a user-level application reading the message from the queue, or of an administrative action, such as deleting the message or the queue. The receipt can represent a positive or negative acknowledgment. Commitment receipts correspond to **final acknowledgments**, as described in [MS-MQQB] section 1.3.5.1.
- **Stream Receipt**: This message acknowledges in-order receipt of stream messages. This receipt is required to guarantee EOIO delivery of stream messages. **Stream receipts** correspond to **order acknowledgments**, as described in [MS-MQQB] section 1.3.5.1.

1.3.9 Protocol Security

The SRMP uses HTTP over Secure Sockets Layer (HTTPS) for secure transport of messages.

1.3.10 WS-Routing (SOAP-RP)

The Web Services Routing Protocol (WS-Routing, formerly known as the SOAP Routing Protocol [SOAP-RP]) is a SOAP-based, stateless protocol for exchanging one-way SOAP messages from an initial sender to the ultimate receiver, potentially through a set of intermediaries. In addition, WS-Routing provides an optional reverse message path that enables two-way message exchange patterns, such as request/response, peer-to-peer conversations, and the return of message acknowledgments and faults. WS-Routing is expressed as a SOAP header entry within a SOAP envelope, making it relatively independent of the underlying protocol.

The SRMP utilizes several fields of the WS-Routing specification and ignores others. In particular, the actual routing-related fields are largely ignored. Instead, the SRMP relies on HTTP and TCP/IP routing between the sender and the receiving Web service. This document describes the WS-Routing fields that the SRMP uses. For more information on WS-Routing, see [MSDN-WSROUTING].

1.3.11 Unicast vs. Multicast Messages

SRMP messages can be sent to either a single destination queue (called a unicast message) or to multiple destination queues simultaneously (called a multicast message). If unicast messaging is used, the underlying transport protocol is HTTP 1.1. If multicast messaging is used, the underlying transport protocol is PGM [RFC3208]. The only visible difference in the protocol message is the format name of the destination queue:

- If the message is sent to a single destination queue as a unicast message, the destination queue name is stated as a regular Uniform Resource Identifier (URI) using the HTTP protocol, as follows:

```
<to>http://destinationhost/msmq/private$/simpleq</to>
```

- If the message is sent to multiple destination queues as a multicast message, the destination queue name is stated as a multicast URI using the PGM protocol, as follows:

```
<to>MSMQ:MULTICAST=234.1.1.1:8001</to>
```

1.3.12 SRMP Example Message

The following is a typical SRMP message that illustrates the basic message structure. At the top is the HTTP POST request. This is followed by the MIME content type information. The SOAP 1.1 message begins after the SOAP boundary data.

In the SOAP message, the SOAP <Envelope> element is the top-level element, immediately followed by the SOAP <Header> and <Body> elements. The <Body> element is empty and ignored by SRMP. Any user messages to be transferred to the receiving application are attached in the form of multipart MIME attachments.

In the <Header> element, the WS-Routing <Path> element is used for addressing the message, specifying the destination queue, and the sending QM's ID. This is followed by the SRMP <Properties> elements, such as <expiresAt>, followed by MSMQ-specific elements, such as <Priority>.

```
POST /msmq/private$/simpleq HTTP/1.1
Host: machine2
Content-Type: multipart/related; boundary="MSMQ - SOAP boundary, 53287";
type=text/xml
Content-Length: 906
SOAPAction: "MSMQMessage"
Proxy-Accept: NonInteractiveClient

--MSMQ - SOAP boundary, 53287
Content-Type: text/xml; charset=UTF-8
Content-Length: 619

<se:Envelope xmlns:se="http://schemas.xmlsoap.org/soap/envelope/"
xmlns="http://schemas.xmlsoap.org/srmp/">
  <se:Header>
    <path xmlns="http://schemas.xmlsoap.org/rp/" se:mustUnderstand="1">
      <action>MSMQ:mqsender label</action>
      <to>http://machine2/msmq/private$/simpleq</to>
      <id>uuid:1@00000000-0000-0000-0000-000000000000</id>
    </path>
    <properties se:mustUnderstand="1">
      <expiresAt>20070609T164419</expiresAt>
      <sentAt>20070608T164419</sentAt>
    </properties>
  </se:Header>
  <se:Body></se:Body>
</se:Envelope>--MSMQ - SOAP boundary, 53287
Content-Type: application/octet-stream
Content-Length: 13
Content-Id: body@ff3af301-3196-497a-a918-72147c871a13

First Message--MSMQ - SOAP boundary, 53287--
```

1.3.13 Typical Message Queuing Scenario

A typical message queuinq scenario achieves reliable, asynchronous messaging between a client computer and a server application. The client application might be an order application used for entering orders from customers. This application could be installed on a laptop computer, moving with the salesperson from customer site to customer site. Connectivity might not be available from those customer sites to the head office. In these cases, the order application on the salesperson's laptop computer would use message queuinq to queue messages containing order information to a local outgoing queue on the laptop computer.

When the salesperson returns to the branch office, the salesperson establishes connectivity with the head office, and the queued messages are then transferred using the SRMP from the local message queue on the laptop computer to a Web service running on the message queue server at the head office. The Web service passes the SRMP messages on to the QM on the server, which places them into the receiving message queue. At that point the orders are retrieved from the message queue on the server and processed by the server application.

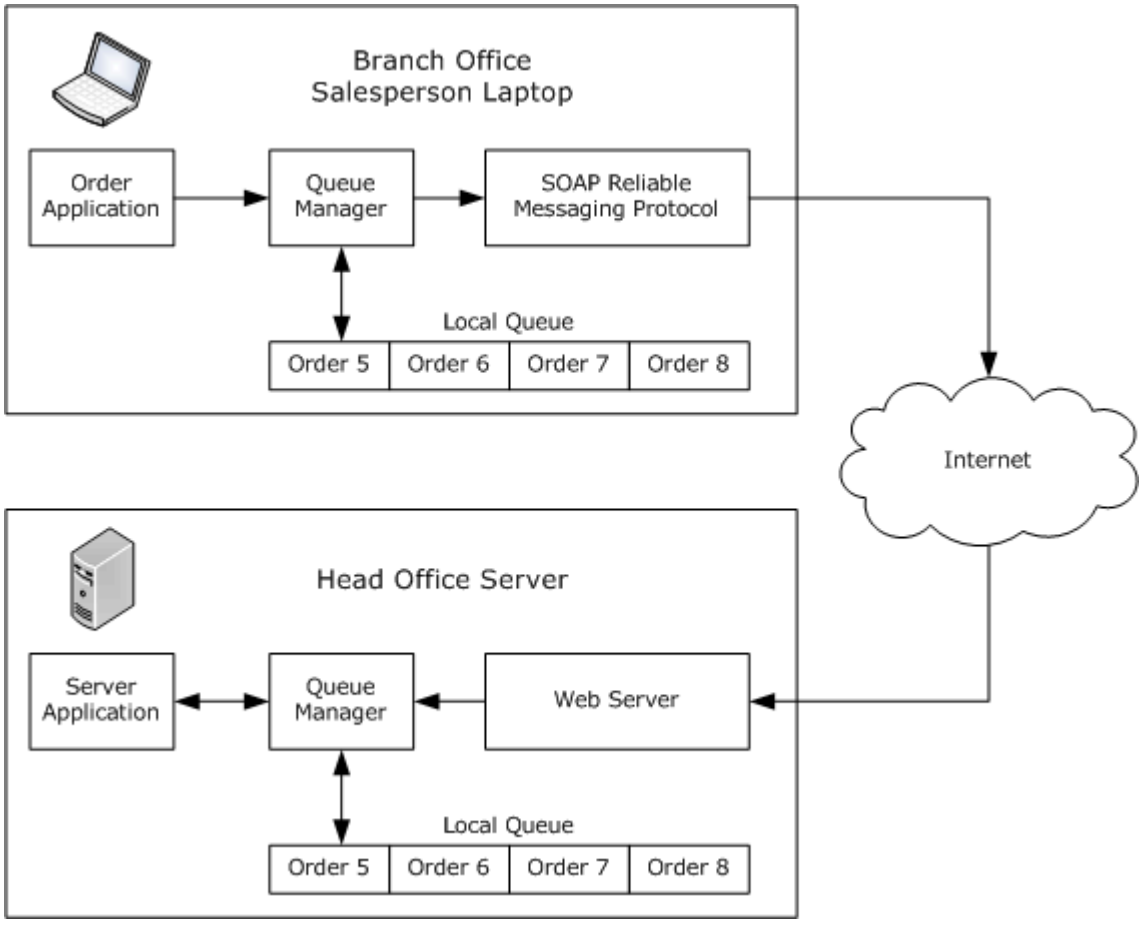

Figure 1: A typical message queuing scenario

1.4 Relationship to Other Protocols

The SRMP depends on SOAP 1.1 and HTTP 1.1 to provide a transport for messages. In the point-to-point scenario, HTTP utilizes direct TCP to transport the message; in the multicast scenario, HTTP utilizes PGM to transport the message.

The SRMP replaces the Message Queuing Binary Protocol, as specified in [MS-MQQB], and provides equivalent functionality in Internet settings. Figure 2 shows a diagram of the protocol layers.

Figure 2: Protocol layer diagram

1.5 Prerequisites/Preconditions

It is assumed that, before invoking SRMP, the protocol client has obtained the name of a server computer that supports this protocol and the name of a queue hosted on the server. This specification does not mandate how a client acquires this information, which typically occurs during the interaction between a client application and the QM's API.

1.6 Applicability Statement

The implementation of the server side of this protocol applies to QMs that provide message queuinq communication services to clients. The implementation of the client side of this protocol applies to client libraries that provide message QMs to applications, or to QMs that delegate requests on behalf of a client.

Applicable scenarios include cases in which users are disconnected or in which connectivity is unreliable, such as when a sales force works remotely, or cases in which guaranteed delivery is important, such as when sending orders from an entry system to a billing system.

Client applications should use SRMP instead of the Message Queuing Binary Protocol, as specified in [MS-MQQB], whenever communication between message queues is over long distance, that is, in Internet scenarios.

The SRMP does not apply in the following scenarios:

- To distributed applications that require message delivery within a predefined amount of time.
- In situations that require message data greater than 4 MB in size.

- If the server side is not running a Web server.

1.7 Versioning and Capability Negotiation

This document covers versioning issues in the following areas:

- **Supported Transports**: This protocol is implemented on top of HTTP 1.1, as discussed in section 2.1.
- **Protocol Versions**: There is a single version of this protocol.
- **Capability Negotiation**: There are no capabilities to negotiate.

1.8 Vendor-Extensible Fields

This protocol does not define any vendor-extensible fields.

1.9 Standards Assignments

This protocol does not utilize any standardized assignments.

2 Messages

2.1 Transport

The SRMP MUST be carried out over SOAP 1.1 [SOAP1.1] and HTTP 1.1 [RFC2616]. Both the sending and the receiving sides MUST provide Web services with the following capabilities:

- MUST support SOAP (as specified in [SOAP1.1]) over HTTP 1.1 (as specified in [RFC2616]) over TCP/IP.
- SHOULD support HTTPS for securing communication with clients.
- SHOULD support PGM (as specified in [RFC3208]) as an alternative to TCP for multicast messages.<1>

Each Web service MUST expose the following TCP ports as endpoints for the HTTP over TCP/IP transport:

- **Port 80**: The default HTTP port.
- **Port 443**: The default HTTPS port for secure communication using Secure Sockets Layer (SSL) [RFC2818].

When using PGM as the transport, both the multicast address and the port must be agreed upon. The mechanism for determining the multicast address and the port are outside the scope of this document.

2.1.1 Use of PGM

The PGM specification [RFC3208] is ambiguous in a number of areas. SRMP's use of PGM is specified below. All sections of [RFC3208] except sections 7, 9.5, and 11-15 MUST be implemented (that is, no Network Element or Designated Local Repairer functionality, nor Appendices A-E).<2>

2.1.1.1 Clarifications on RFC3208

The following values SHOULD<3> be used for the constants defined in [RFC3208]:

Constant	Value
TXW_MAX_RTE	70 kBps
TXW_SECS	300
TXW_ADV_SECS	15% of TXW_SECS
TXW_ADV_IVL	15% of TXW_SECS
IHB_MIN	1 second
IHB_MAX	15 seconds
NAK_RPT_IVL	0.75 secs
NAK_RDATA_IVL	2 secs
NAK_NCF_RETRIES	10

Constant	Value
NAK_DATA_RETRIES	10
Token bucket size	40 msecs

In addition, [RFC3208] leaves up to implementations several other details, which are clarified as follows:

- The NAK_RB_IVL timer SHOULD be chosen randomly from the interval [0.05, 0.01] seconds.
- The SPM ambient time interval MUST be implemented such that ambient SPMs are sent when either 50 data (ODATA or RDATA) packets have been transmitted, or 0.5 seconds have passed since the last ambient SPM, whichever comes sooner.
- Section 5.3 of [RFC3208] allows a source to delay RDATA retransmission to accommodate the arrival of additional NAKs. An implementation of this specification SHOULD delay such retransmissions by a time computed according to the following formula:

```
RDataDelayTime =
((RDataSequenceNumber - TrailingSequenceNumber) * 60 msecs) /
((LastODataSentSequenceNumber - TrailingSequenceNumber + 1))
```

- Section 16 of [RFC3208] allows implementations to implement any scheme for advancing the transmit window. Implementations of this specification SHOULD advance the transmit window every TXW_ADV_IVL.
- Implementations of this specification SHOULD delay transmit window advancement if the sender has pending NAK requests in the range of sequences that the trailing edge of the window was supposed to advance over; in this case, the trailing edge will only be advanced up to the first pending NAK request.

2.2 Message Syntax

This section specifies the syntax of SOAP 1.1 [SOAP1.1] messages that are exchanged using the SRMP. All messages are encoded in XML [XML1.0], as specified in [SOAP1.1] section 3.

2.2.1 Common Data Types

2.2.1.1 GUID String

This type is a string representation of a **GUID** type, as specified in [MS-DTYP] section 2.3.2.1, in the string form of a **UUID**, as specified in [RFC4122] section 3.

2.2.1.2 ISO 8601 Date String

This string is an ISO 8601 [ISO-8601] formatted date and time using the format "YYYYMMDDThhmmss".

2.2.1.3 ISO 8601 Duration String

This string is an ISO 8601 [ISO-8601] formatted duration using the format "PnnYnnMnnDTnnHnnMnnS".

2.2.1.4 xs:unsignedLong

This is an unsigned long integer, as described in [W3C-XSD].

2.2.2 SRMP Message Structure

All SRMP messages MUST conform to the basic structure of a SOAP 1.1 message, as specified in [SOAP1.1] section 4, as follows:

- The SOAP Envelope element <se:Envelope>, as defined in namespace http://schemas.xmlsoap.org/soap/envelope, MUST be present as the top-level element of the SRMP message.
- The SOAP Header element <se:Header>, as defined in namespace http://schemas.xmlsoap.org/soap/envelope, MUST be present as the first immediate child element of the SOAP Envelope element.
- The SOAP Body element <se:Body>, as defined in http://schemas.xmlsoap.org/soap/envelope, MUST be present and MUST be an immediate child element of a SOAP Envelope element. It MUST directly follow the SOAP Header element. The <se:Body> element MUST be empty and MUST be ignored.

The application message payload that the QM on the receiving computer delivers to the receiving application MUST be encoded as a multipart MIME attachment [RFC2387] by the sending QM.

Internal receipt messages (delivery, commitment, and stream receipts) MUST NOT contain a multipart MIME payload attachment.

2.2.3 Standard XML Namespaces

The following table shows the standard XML namespaces used within this protocol and the alias (prefix) used in the remaining sections of this protocol specification. Typically, SRMP messages declare the SOAP envelope and SRMP namespaces in the <se:Envelope> element. The WS-Routing namespace is declared in the <rp:path> element, which is the only element that uses it. The MSMQ namespace is declared in the <msmq:Msmq> element, which is the only element that uses it.

Alias (prefix)	XML namespace	Note
se	http://schemas.xmlsoap.org/soap/envelope/	Standard SOAP envelope namespace.
rp	http://schemas.xmlsoap.org/rp/	Standard WS-Routing namespace.
srmp	http://schemas.xmlsoap.org/srmp/	SRMP namespace; XML schema currently not defined.
msmq	msmq.namespace.xml	Internal MSMQ namespace; used to identify MSMQ elements.

Note that while the SRMP XML namespace uses a URI to define that namespace, the schema for SRMP has not been defined and there are no plans to define it.

A schema for the MSMQ namespace has not been defined and there are no plans to define it.

This document follows the convention that, when discussing elements, they are prefixed with the previously defined aliases unless the context makes it clear which namespace the element is from.

2.2.4 WS-Routing Path Element

The SRMP uses the WS-Routing <rp:path> element, as defined in http://schemas.xmlsoap.org/rp, for its addressing purposes. The <path> element specifies the destination queue for the SRMP message, the sender's identity, a user-defined label for the message, and a response queue.

The WS-Routing element <rp:path>, as defined in namespace http://schemas.xmlsoap.org/rp, MUST be present as a child element of the <se:Header> element. Its namespace MUST be declared as xmlns="http://schemas.xmlsoap.org/rp/". It MUST be marked with the SOAP attribute se:mustUnderstand="1". For example:

```
<path xmlns="http://schemas.xmlsoap.org/rp/" se:mustUnderstand="1">
```

The child elements of <rp:path> are used as described in the following sections.

2.2.4.1 action Element

The WS-Routing <action> element MUST be present as a child element of the <rp:path> element. It MAY contain a string representing a user-defined label for the SRMP message (to identify the message to the application). The user-defined label MUST be prefixed by the string "MSMQ:".<4>

For example:

```
<action>MSMQ:mqsender label</action>
```

If the message is a delivery receipt (see section 2.2.5.4) or a commitment receipt (see section 2.2.5.5) in response to a previous SRMP message (containing <deliveryReceiptRequest> or <commitmentReceiptRequest>), the <action> element MUST contain the identical label string as the initial message.<5>

If the message is a stream receipt (see section 2.2.5.6) in response to a previous stream message, the <action> element MUST contain the label string "MSMQ:QM Ordering Ack".

The <action> element corresponds to the **MessagePropertiesHeader.MessageData.Label** field of the Message Queuing Binary Protocol (for details, see [MS-MQQB] section 2.2.2.4).

2.2.4.2 to Element

The WS-Routing <to> element MUST be present as a child element of the <rp:path> element. It MUST contain a string representing the URI of the destination queue (see [MS-MQQB] section 2.2.2.3) that is the message's ultimate destination. For details on MSMQ naming, see [MS-MQMQ] section 2.1.

If HTTP is selected as the underlying transport protocol, the URI of the destination queue MUST begin with "http://". For example:

```
<to>http://myhostname/msmq/private$/sampleq</to>
```

If HTTPS is selected as the underlying transport protocol, the URI of the destination queue MUST begin with "https://". This is how transport security is activated for SRMP messages. For example:

```
<to>https://myhostname/msmq/private$/sampleq</to>
```

If PGM [RFC3208] is selected as the underlying transport protocol, the URI of the destination queue MUST begin with "MSMQ:MULTICAST=", followed by the multicast IP address of the destination queue, followed by ":", followed by the port number of the destination queue. This is how multicast transport is activated for SRMP messages. For example:

```
<to>MSMQ:MULTICAST=234.1.1.1:8001</to>
```

2.2.4.3 id Element

The WS-Routing <id> element MUST be present as a child element of the <rp:path> element. The content of this element uniquely identifies the message across all MSMQ QMs.

If the <Msmq> element is not present in the message (see section 2.2.6), the <id> element MUST be ignored.

If the <Msmq> element is present in the message, the <id> element MUST contain a string beginning with "uuid:", followed by an index number of type xs:unsignedLong identifying the message, followed by "@", followed by a GUID String identifying the sending QM.

The sending QM GUID SHOULD<6> be identical to the <SourceQmGuid> child element of the <Msmq> element.

The message index corresponds to the **UserHeader.MessageID** field in the Message Queuing Binary Protocol (for more information, see [MS-MQQB] section 2.2.2.3). The GUID of the QM corresponds to the **UserHeader.SourceQueueManager** field in the Message Queuing Binary Protocol (for more information, see [MS-MQQB] section 2.2.2.3).

For example:

```
<id>uuid:26626@32221eda-9376-46df-b6ed-783091123831</id>
```

2.2.4.4 rev Element

The WS-Routing <rev> element MAY be present as a child element of the <rp:path> element. If present, exactly one<7> WS-Routing <via> element MUST be present as a child element of the <rev> element.

The <rev> element describes the reverse path that reverse path messages must follow. This element SHOULD contain a string representing the URI of the application-level response queue (see [MS-MQQB] section 2.2.2.3) in the <via> element. The URI MAY be an HTTP/HTTPS format name (similar to the URI in the <to> element described in section 2.2.4.2), or it MAY be a standard format name prefixed with "MSMQ:" For details on MSMQ naming, see [MS-MQMQ] section 2.1.

The <via> element MAY be empty,<8> which is equivalent to not including a <rev> element in the message.

For example:

```
<rev>
    <via>http://myhostname/msmq/private$/sampleq</via>
</rev>
```

2.2.4.5 Other Elements

The WS-Routing elements <fwd>, <from>, <relatesTo>, and <fault> MUST NOT be present as child elements of the <rp:path> element and MUST be ignored.

2.2.5 SRMP Header Elements

The SRMP defines six protocol-specific elements inside the SOAP header: <properties>, <services>, <stream>, <deliveryReceipt>, <commitmentReceipt>, and <streamReceipt>. The SRMP namespace MUST be declared as xmlns="http://schemas.xmlsoap.org/srmp/", either globally in the <se:Header> element or locally in each SRMP element.

2.2.5.1 properties Element

The SRMP <properties> element MUST be present as a child element of the <se:Header> element. It MUST be marked with the SOAP attribute se:mustUnderstand="1". This element specifies common message properties of SRMP messages. For example:

```
<properties se:mustUnderstand="1">
```

The child elements of the <properties> element are used as described in the following sections.

2.2.5.1.1 expiresAt Element

The SRMP <expiresAt> element MUST be present as a child element of the <properties> element. It MUST contain an ISO 8601 Date String and is expressed in **Coordinated Universal Time (UTC)**.

The <expiresAt> element represents the expiration time stamp of the message, beyond which the message MUST not be processed by the receiver and MUST be discarded. For example:

```
<expiresAt>20070619T210654</expiresAt>
```

2.2.5.1.2 duration Element

The SRMP <duration> element MAY be present as a child element of the <properties> element. It MUST contain an ISO 8601 Duration String.

The <duration> element represents the amount of time remaining relative to the current time, beyond which time the message MUST NOT be processed by the receiver and MUST be discarded. For example: "PnnYnnMnnDTnnHnnMnnS".

```
<duration>PT12H</duration>
```

2.2.5.1.3 sentAt Element

The SRMP <sentAt> element MAY be present as a child element of the <properties> element.

It MUST contain an ISO 8601 Date String and is expressed in UTC.

The <sentAt> element represents the sending time stamp of the message. It MUST NOT be modified on any attempt at retransmission of the message if previous transmission attempts have failed. For example:

```
<sentAt>20070618T210654</sentAt>
```

2.2.5.1.4 inReplyTo Element

The SRMP <inReplyTo> element MAY be present as a child element of the <properties> element and MUST be ignored.

2.2.5.2 services Element

The SRMP <services> element MAY be present as a child element of the <se:Header> element. If present, it MUST be marked with the SOAP attribute se:mustUnderstand="1".

This element specifies services related to the delivery guarantees of SRMP messages. For example:

```
<services se:mustUnderstand="1">
```

The child elements of <services> are used as described in the following sections.

2.2.5.2.1 durable/ Element

The SRMP <durable/> element MAY be present as a child element of the <services> element; if the <stream> element (see section 2.2.5.3) is present, <durable/> MUST be present.

If <durable/> is present, the message MUST be persisted to stable storage at the sender before transmission, and MUST be persisted at the receiver immediately on receipt prior to further processing.

If a delivery receipt has been requested (see section 2.2.5.2.2), the receipt MUST be sent only after the message has been durably stored. For example:

```
<durable/>
```

2.2.5.2.2 deliveryReceiptRequest Element

The SRMP <deliveryReceiptRequest> element MAY be present as a child element of the <services> element. If present, the SRMP <sendTo> element MUST be present as a child element of the <deliveryReceiptRequest> element. The <sendTo> element MUST contain a string representing the URI of the administration queue that receipts should be sent to. The URI MUST be an HTTP/HTTPS format name (similar to the URI in the <to> element described in section 2.2.4.2).

If <deliveryReceiptRequest> is present, the receiver MUST acknowledge acceptance of the message with a receipt. The receipt acknowledges that the receiver has understood each SRMP element

marked with "mustUnderstand=1", performed all actions required of the SRMP receiver prior to sending the receipt, and is committed to performing all actions required of the receiver after sending the receipt. In the context of MSMQ, this means that the message was received and placed in the destination queue at the receiver. The receipt MUST be sent to the URI specified in the <sendTo> element. For example:

```
<deliveryReceiptRequest>
    <sendTo>http://myhostname/MSMQ/private$/receipts</sendTo>
</deliveryReceiptRequest>
```

2.2.5.2.3 commitmentReceiptRequest Element

The SRMP <commitmentReceiptRequest> element MAY be present as a child element of the <services> element. If present:

- The SRMP <sendTo> element MUST be present as a child element of the <commitmentReceiptRequest> element. The <sendTo> element MUST contain a string representing the URI of the admin queue that receipts should be sent to. The URI MUST be an HTTP/HTTPS format name (similar to the URI in the <to> element described in section 2.2.4.2).
- The SRMP <positiveOnly/> element MAY be present as a child element of the <commitmentReceiptRequest> element. If present, the receiver MUST send a positive commitment receipt if it has understood all elements marked with "mustUnderstand=1" and is committed to processing the message. In the context of MSMQ, this means that the message was received, placed in the destination queue at the receiver, and successfully removed from the destination queue by the receiving application.
- The SRMP <negativeOnly/> element MAY be present as a child element of the <commitmentReceiptRequest> element. If present, the receiver MUST send a negative commitment receipt if it attempted to process the message, but decided not to commit to complete its processing; or the message expired before the receiver attempted to process the message. In the context of MSMQ, this means that the message was discarded or otherwise removed from the destination queue before the receiving application was able to pick it up.
- If neither <positiveOnly/> nor <negativeOnly/> is present, the receiver MUST NOT send any commitment receipts.
- If both <positiveOnly/> and <negativeOnly/> are present, the receiver MUST send a positive and negative commitment receipt, as described above.

For example:

```
<commitmentReceiptRequest>
    <sendTo>http://myhostname/MSMQ/private$/deliveryDone</sendTo>
    <positiveOnly/>
</commitmentReceiptRequest>
```

2.2.5.3 stream Element

The SRMP <stream> element MAY be present as a child element of the <se:Header> element. If present, it MUST be marked with the SOAP attribute se:mustUnderstand="1".

If a <stream> element is present, the <Msmq> element specified in section 2.2.6 MUST be present. Messages sent in a stream MUST be persistent; therefore, the <durable/> child element of the <services> element MUST be present (see section 2.2.5.2.1).

The <stream> element defines child elements for a label and context for message **sequences**. In particular, it defines the concepts of the first message, current message, and previous message. The receiver MUST NOT process any message more than once, and the receiver MUST NOT accept any message unless the previous message has been accepted. For example:

```
<stream se:mustUnderstand="1">
```

The child elements of <stream> are used as described in the following sections.

2.2.5.3.1 streamId Element

The SRMP <streamId> element MUST be present as a child element of the <stream> element. It defines a unique identifier for the stream and MUST begin with "uid:", followed by a GUID String that identifies the sending QM, followed by a backslash (\), followed by a sequence number of type xs:unsignedLong.

The <streamId> element corresponds to a **MessageIdentifier** and **MessageIdOrdinal**, as specified in [MS-MQQB]. For example:

```
<streamId>uid:4a85b192-3ccd-4ba2-a0ac-7f0a11be1b08\111</streamId>
```

2.2.5.3.2 current Element

The SRMP <current> element MUST be present as a child element of the <stream> element.

It MUST contain a number of type xs:unsignedLong defining the relative location of the message in the stream. The first message in the stream MUST be assigned the value 1. For example:

```
<current>42</current>
```

2.2.5.3.3 previous Element

The SRMP <previous> element MAY be present as a child element of the <stream> element.

If present, it MUST contain a number of type xs:unsignedLong defining the relative location of the preceding message in the stream.

The <previous> element is used to indicate gaps in the stream that may occur if the sender decides to skip some messages without invalidating the entire stream. For example:

```
<previous>41</previous>
```

2.2.5.3.4 start Element

The SRMP <start> element MAY be present as a child element of the <stream> element. If present, the SRMP element <sendReceiptsTo> MUST be present as a child element of the <start> element. The <sendReceiptsTo> element MUST contain a string representing the URI of the administration

queue that receipts should be sent to. The URI MUST be an HTTP/HTTPS format name (similar to the URI in the <to> element described in section 2.2.4.2).

The <start> element MUST be present in the first message of a stream and MUST NOT be present in any subsequent message of the stream. For example:

```
<start>
    <sendReceiptsTo>
        http://myhostname/MSMQ/private$/receipts
    </sendReceiptsTo>
</start>
```

2.2.5.4 deliveryReceipt Element

The SRMP <deliveryReceipt> element MAY be present as a child element of the <se:Header> element. This element MUST be generated in response to a message that contains the <deliveryReceiptRequest> element (see section 2.2.5.2.2). If present:

- The SRMP element <receivedAt> MUST be present as a child element of the <deliveryReceipt> element. It MUST contain a string representing an ISO 8601 Date String. The <receivedAt> element represents the current time stamp at which the receiving QM placed the message in its destination queue.
- The SRMP <id> element MUST be present as a child element of the <deliveryReceipt> element. It MUST contain the same MessageID as the <id> element in the <path> element of the original message that requested the delivery receipt (see section 2.2.4.3).
- The <to> element of the <path> element in the receipt message MUST contain the URI of the administration queue specified in the <sendTo> element of the <deliveryReceiptRequest> in the original message (see section 2.2.5.2.2).

For example:

```
<deliveryReceipt>
    <receivedAt>20070618T210655</receivedAt>
    <id>uuid:32852@32221eda-9376-46df-b6ed-783091123831</id>
</deliveryReceipt>
```

2.2.5.5 commitmentReceipt Element

The SRMP <commitmentReceipt> element MAY be present as a child element of the <se:Header> element. This element MUST be generated in response to a message that contains the <commitmentReceiptRequest> element (see section 2.2.5.2.3). If present:

- The SRMP element <decidedAt> MUST be present as a child element of the <commitmentReceipt> element. It MUST contain a string representing an ISO 8601 Date String and is expressed in UTC. The <decidedAt> element represents the current time stamp at which the message was removed from the destination queue, either by the receiving application (in the positive case) or by a discard or purge operation (in the negative case).
- The SRMP <decision> element MUST be present as a child element of the <commitmentReceipt> element. It MUST contain one of two enumeration values: "negative" or "positive", corresponding to negative or positive commitment receipts.

- The SRMP <id> element MUST be present as a child element of the <commitmentReceipt> element. It MUST contain the same MessageID as the <id> element in the <path> element of the original message that requested the commitment receipt (see section 2.2.4.3).
- The <to> element of the <path> element in the receipt message MUST contain the URI of the administration queue specified in the <sendTo> element of the <commitmentReceiptRequest> in the original message (see section 2.2.5.2.3).

For example:

```
<commitmentReceipt>
    <decidedAt>20070618T210908</decidedAt>
    <decision>positive</decision>
    <id>uuid:32852@32221eda-9376-46df-b6ed-783091123831</id>
</commitmentReceipt>
```

2.2.5.6 streamReceipt Element

The SRMP <streamReceipt> element MAY be present as a child element of the <se:Header> element. This element MUST be generated in response to each message that contains the <stream> element (see section 2.2.5.3). If present:

- The SRMP <streamId> element MUST be present as a child element of the <streamReceipt> element. It MUST contain the same StreamID as the <streamId> element in the <stream> element of the original stream message received (see section 2.2.5.3.1).
- The SRMP <lastOrdinal> element MUST be present as a child element of the <streamReceipt> element. It MUST contain a number of type xs:unsignedLong identical to the <current> element of the last stream message received. By sending this stream receipt, the receiver acknowledges receipt of all messages in the stream up to and including the message whose <current> number is identical to the <lastOrdinal> element.

For example:

```
<streamReceipt>
    <streamId>uid:4a85b192-3ccd-4ba2-a0ac-7f0a11be1b08\111</streamId>
    <lastOrdinal>2</lastOrdinal>
</streamReceipt>
```

2.2.6 MSMQ Elements

MSMQ uses the following XML elements for specific purposes. They each have corresponding fields in the Message Queuing Binary Protocol [MS-MQQB].

MSMQ elements are child elements of the <Msmq> element. The <Msmq> element MAY be present as a child element of the <se:Header> element. If present, the MSMQ namespace MUST be declared as xmlns="msmq.namespace.xml". For example:

```
<Msmq xmlns="msmq.namespace.xml">
```

The child elements of <Msmq> are used as described in the following sections.

2.2.6.1 Class Element

The MSMQ <Class> element MUST be present as a child element of the <Msmq> element. The values in this element directly match those listed in [MS-MQQB] section **2.2.1.5**. The <Class> element MUST contain a number of type xs:unsignedLong. For example:

```
<Class>255</Class>
```

2.2.6.2 Priority Element

The MSMQ <Priority> element MUST be present as a child element of the <Msmq> element. The <Priority> element directly maps to the **BaseHeader.Flags.PR** field described in [MS-MQQB]. It MUST contain a number of type xs:unsignedLong with valid values from 0 to 7. For example:

```
<Priority>3</Priority>
```

2.2.6.3 Journal/ Element

The MSMQ <Journal/> element MAY be present as a child element of the <Msmq> element. The <Journal/> element maps directly to the **UserHeader.Flags.JP** flag described in [MS-MQQB]. For example:

```
<Journal/>
```

2.2.6.4 DeadLetter/ Element

The MSMQ <DeadLetter/> element MAY be present as a child element of the <Msmq> element. The <DeadLetter/> element maps directly to the **UserHeader.Flags.JN** flag described in [MS-MQQB]. For example:

```
<DeadLetter/>
```

2.2.6.5 Correlation Element

The MSMQ <Correlation> element MAY be present as a child element of the <Msmq> element. The <Correlation> element maps directly to the **MessagePropertiesHeader.CorrelationId** field. The **MessagePropertiesHeader.CorrelationId** MUST be encoded as a set of bytes in base64, as described in [RFC3548]. For example:

```
<Correlation>AAAAAAAAAAAAAAAAAAAAAAAAAA=</Correlation>
```

2.2.6.6 Trace/ Element

The MSMQ <Trace/> element MAY be present as a child element of the <Msmq> element. The <Trace/> element maps directly to the **BaseHeader.Flags.TR** flag described in [MS-MQQB]. For example:

```
<Trace/>
```

2.2.6.7 ConnectorType Element

The MSMQ <ConnectorType> element MAY be present as a child element of the <Msmq> element. It MUST contain a GUID string. The <ConnectorType> element maps directly to the **UserHeader.ConnectorType** as described in [MS-MQQB]. For example:

```
<ConnectorType>fd74b8eb-2af7-4ac5-9405-074e315df392</ConnectorType>
```

2.2.6.8 App Element

The MSMQ <App> element MAY be present as a child element of the <Msmq> element. It MUST contain a number of type xs:unsignedLong. The <App> element maps directly to the **MessagePropertiesHeader.ApplicationTag** field described in [MS-MQQB]. For example:

```
<App>36</App>
```

2.2.6.9 BodyType Element

The MSMQ <BodyType> element MUST be present as a child element of the <Msmq> element. It MUST contain a number of type xs:unsignedLong. The <BodyType> element maps directly to the **MessagePropertiesHeader.BodyType** field described in [MS-MQQB]. For example:

```
<BodyType>8</BodyType>
```

2.2.6.10 HashAlgorithm Element

The MSMQ <HashAlgorithm> element MAY be present as a child element of the <Msmq> element. It MUST contain a number of type xs:unsignedLong. The <HashAlgorithm> directly maps to the **MessagePropertiesHeader.HashAlgorithm** field described in [MS-MQQB]. For example:

```
<HashAlgorithm>32772</HashAlgorithm>
```

2.2.6.11 Eod Element

The MSMQ <Eod> element MAY be present as a child element of the <Msmq> element. If present:

- The MSMQ <First/> element MAY be present as a child element of the <Eod> element. The <First/> element directly maps to the **TransactionHeader.FM** flag described in [MS-MQQB].
- The MSMQ <Last/> element MAY be present as a child element of the <Eod> element. The <Last/> element directly maps to the **TransactionHeader.LM** flag described in [MS-MQQB].
- The MSMQ <ConnectorId> element MAY be present as a child element of the <Eod> element. It MUST contain a GUID string. The <ConnectorId> directly maps to the **TransactionHeader.ConnectorQMGuid** field described in [MS-MQQB].

For example:

```
<Eod>
    <First/>
    <Last/>
</Eod>
```

2.2.6.12 Provider Element

The MSMQ <Provider> element MAY be present as a child element of the <Msmq> element. If present:

1. The MSMQ <Type> element MUST be present as a child element of the <Provider> element. It MUST contain a number of type xs:unsignedLong.
2. The MSMQ <Type> element MUST be present as a child element of the <Provider> element. It MUST contain a number of type xs:unsignedLong.

The MSMQ <Type> element MUST be present as a child element of the <Provider> element. It MUST contain a number of type xs:unsignedLong. For example:

```
<Provider>
    <Type>1234</Type>
    <Name>ProviderName</Name>
</Provider>
```

2.2.6.13 SourceQmGuid Element

The MSMQ <SourceQmGuid> element MUST be present as a child element of the <Msmq> element. It MUST contain a GUID string. The <SourceQmGuid> element directly maps to the **UserHeader.SourceQueueManager** field described in [MS-MQQB]. For example:

```
<SourceQmGuid>fd74b8eb-2af7-4ac5-9405-074e315df392</SourceQmGuid>
```

2.2.6.14 DestinationMqf Element

The MSMQ <DestinationMqf> element MAY be present as a child element of the <Msmq> element. It MUST contain one or more MSMQ format names separated by white space. The <DestinationMqf> element directly maps to the **UserMessage.MultiQueueFormatHeader.Destination** field described in [MS-MQQB]. For example:

```
<DestinationMqf>
    http://Machine1/msmq/private$/SimpleQ
    http://Machine2/msmq/private$/SimpleQ
    http://Machine3/msmq/private$/SimpleQ
</DestinationMqf>
```

2.2.6.15 AdminMqf Element

The MSMQ <AdminMqf> element MAY be present as a child element of the <Msmq> element. It MUST contain one or more MSMQ format names separated by white space. The <AdminMqf>

element directly maps to the **UserMessage.MultiQueueFormatHeader.Administration** field described in [MS-MQQB]. For example:

```
<AdminMqf>
    http://Machine1/msmq/private$/AdminQ
    http://Machine2/msmq/private$/AdminQ
    http://Machine3/msmq/private$/AdminQ
</AdminMqf>
```

2.2.6.16 ResponseMqf Element

The MSMQ <ResponseMqf> element MAY be present as a child element of the <Msmq> element. It MUST contain one or more MSMQ format names separated by white space. The <ResponseMqf> element directly maps to the **UserMessage.MultiQueueFormatHeader.Response** field described in [MS-MQQB]. For example:

```
<ResponseMqf>
    http://Machine1/msmq/private$/ResponseQ
    http://Machine2/msmq/private$/ResponseQ
    http://Machine3/msmq/private$/ResponseQ
</ResponseMqf>
```

2.2.6.17 TTrq Element

The MSMQ <TTrq> element MUST be present as a child element of the <Msmq> element. It MUST contain an ISO 8601 Duration String. This field directly maps to the **BaseHeader.TimeToReachQueue** field described in [MS-MQQB].

The <TTrq> element represents the amount of time remaining relative to the current time, beyond which time the message MUST NOT be processed by the receiver and MUST be discarded. For example: "PnnYnnMnnDTnnHnnMnnS".

```
<TTrq>PT12H</TTrq>
```

3 Protocol Details

3.1 Abstract Data Model

This section describes a conceptual model of possible data organization that an implementation maintains to participate in this protocol. This document is organized in order 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 Protocol State

This is the protocol state section.

3.1.1.1 State Diagrams

This is the state diagrams section.

3.1.1.1.1 Regular and Durable Message State – Sender

The following figure shows the protocol state at the sender for regular and durable SRMP messages.

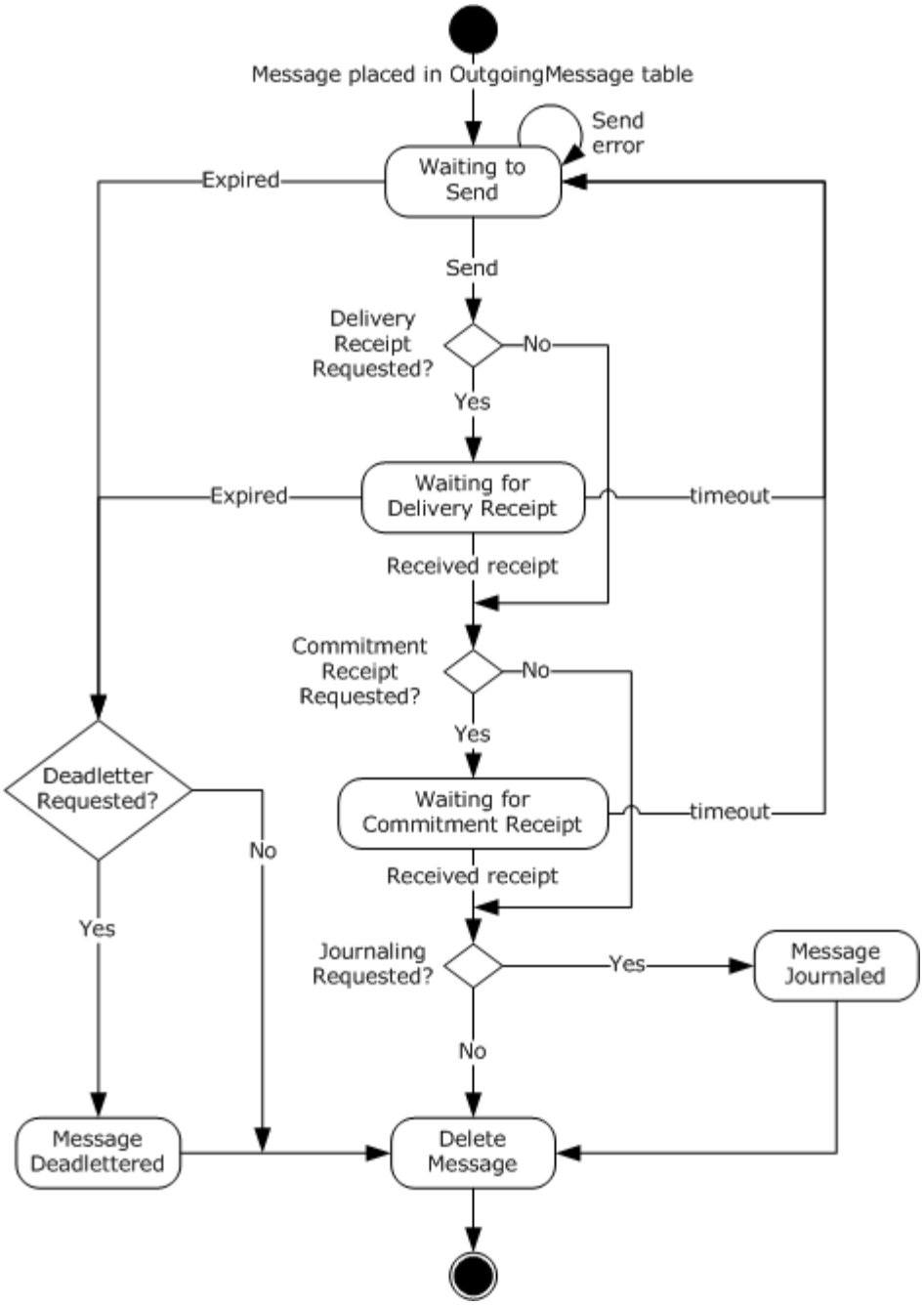

Figure 3: Regular and durable message state - sender

3.1.1.1.2 Regular and Durable Message State – Receiver

The following figure shows the protocol state at the receiver for regular and durable SRMP messages.

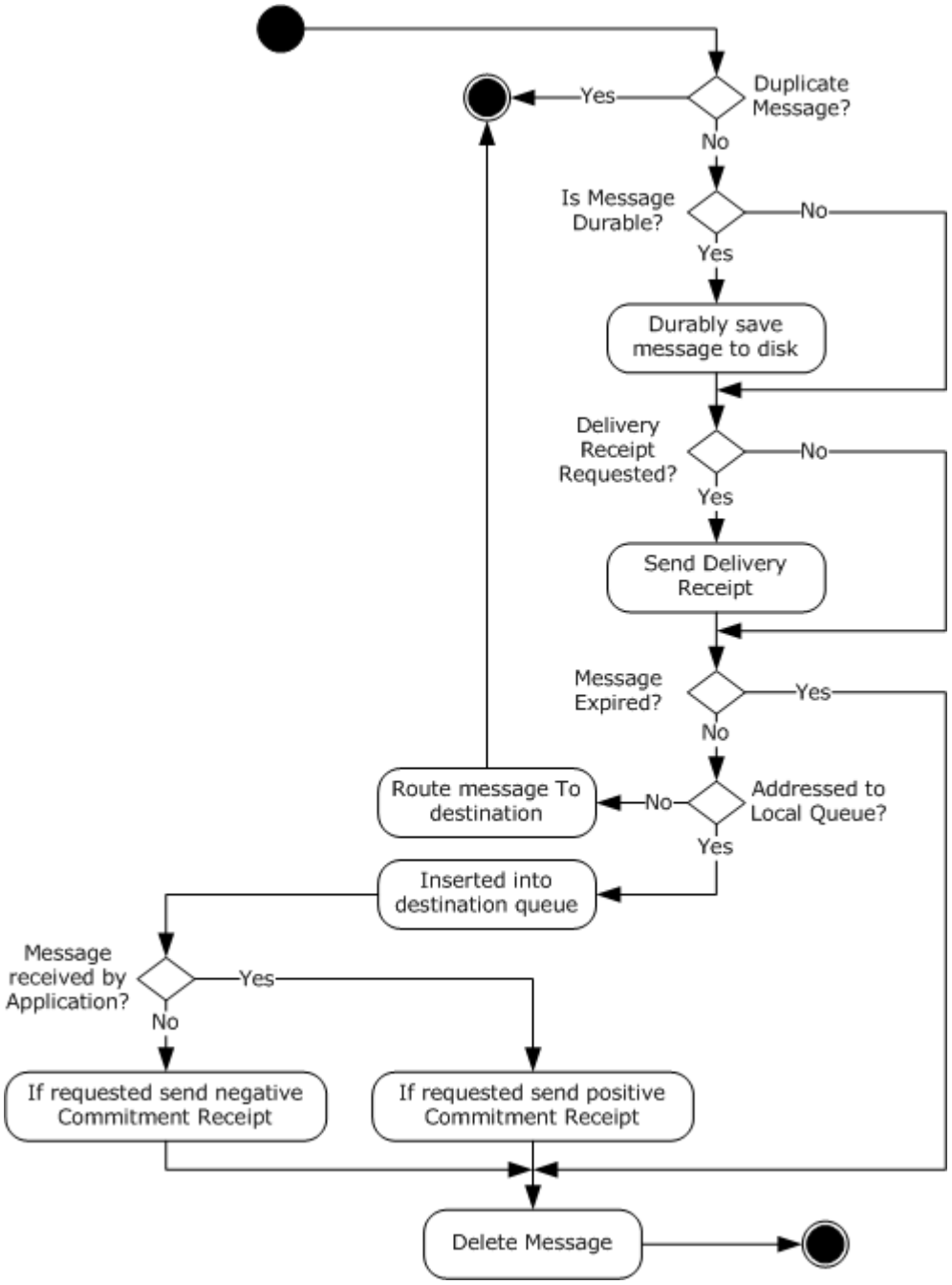

Figure 4: Regular and durable message state - receiver

3.1.1.1.3 Stream Message State – Sender

The following figure shows the protocol state at the sender for SRMP stream messages.

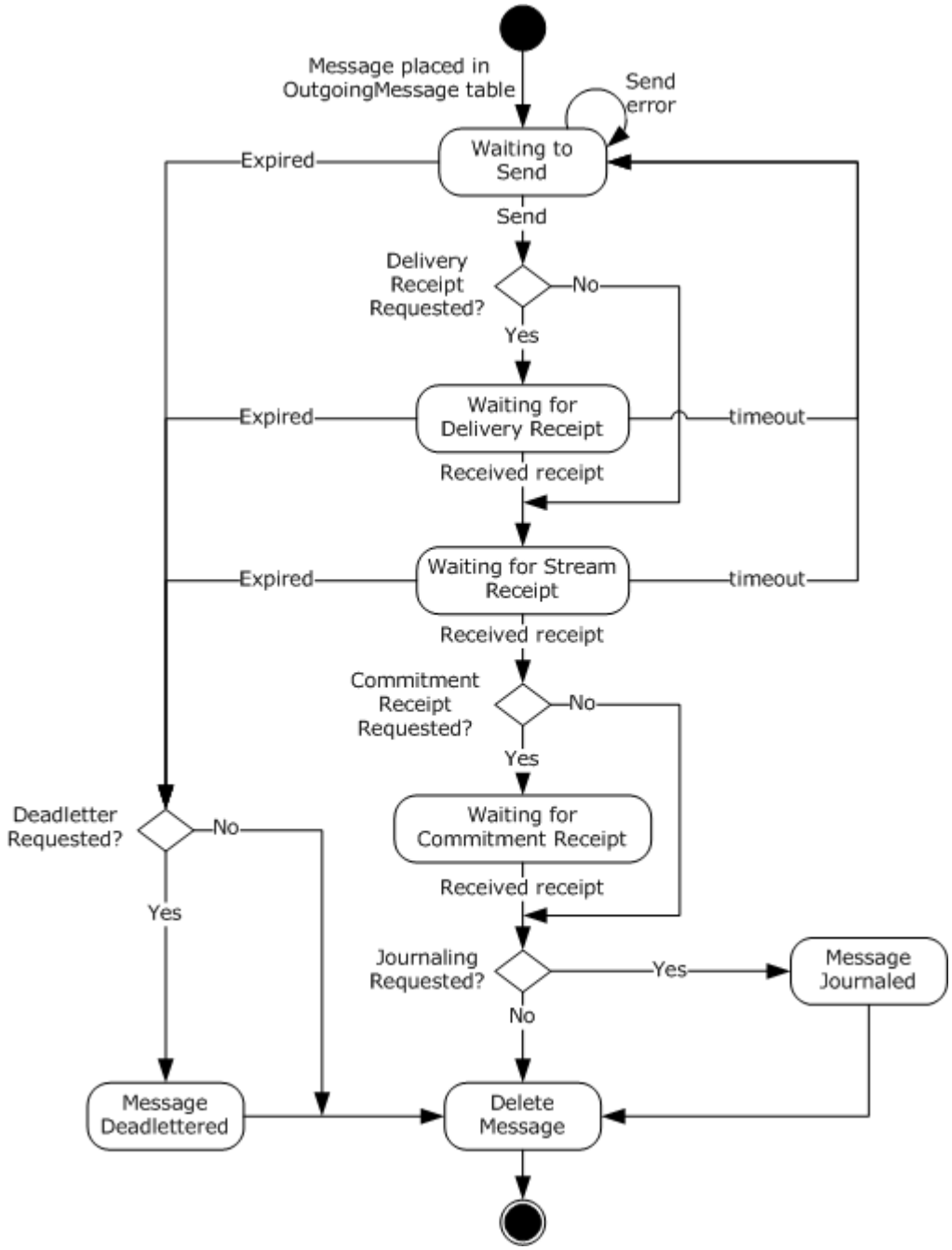

Figure 5: Stream message state - sender

3.1.1.1.4 Stream Message State – Receiver

The following figure shows the protocol state at the receiver for SRMP stream messages.

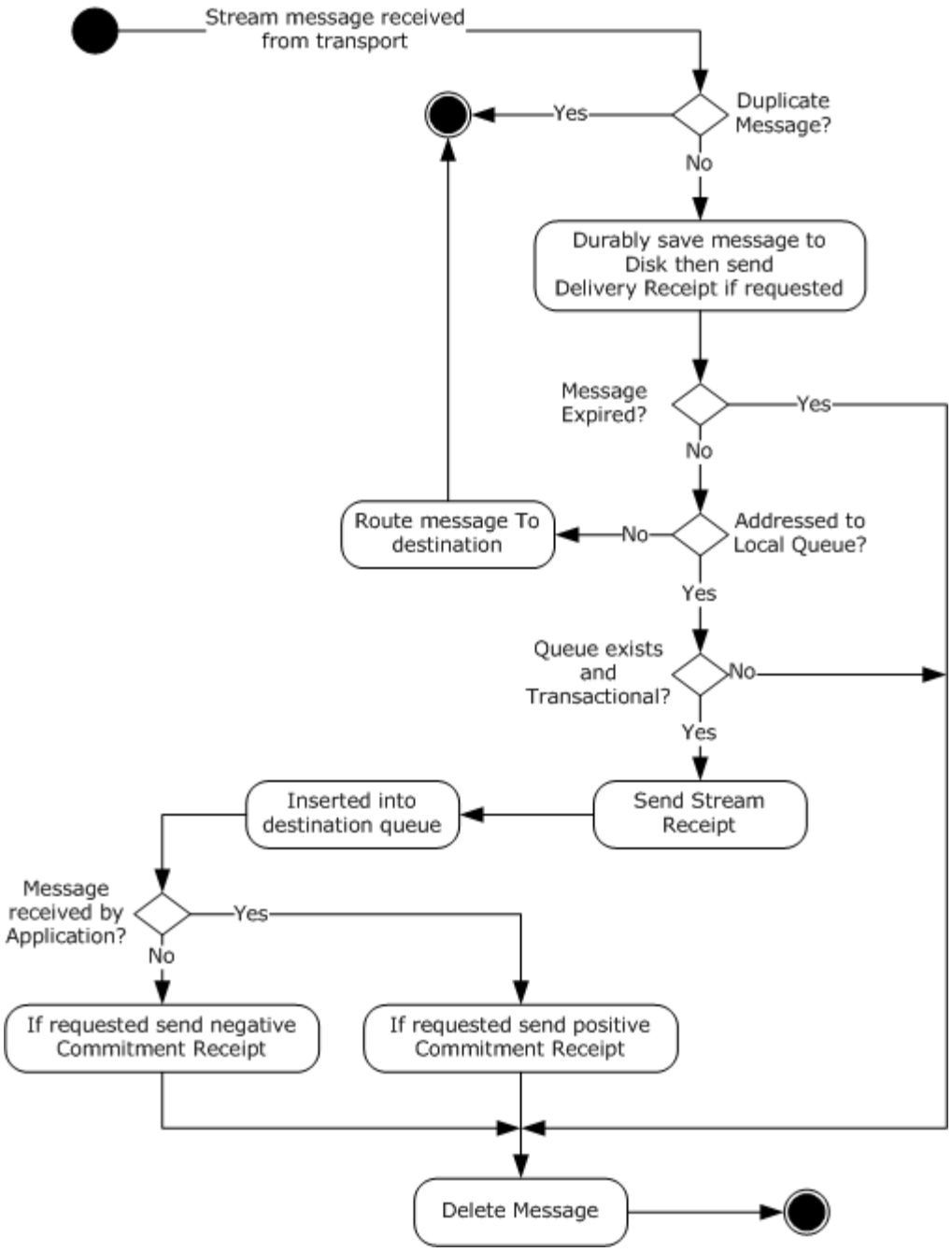

Figure 6: Stream message state - receiver

3.1.1.2 Global State

The SRMP MUST maintain the following global data elements:

- **QMGuid**: The GUID of the local QM. This value uniquely identifies the local host.<9> This value MUST be saved to persistent storage.

- **QueueTable**: A table of queues deployed at the host, keyed by name. Each entry MUST contain:
  - The name of the queue. For more information on queue naming conventions, see [MS-MQMQ] section 2.5.
  - A Boolean transaction flag indicating if the queue supports transactional messages.
  - A message list.
- **Message List**: A list of messages in a queue, in ascending order by arrival time within message priority. Each entry MUST contain:
  - A **MessageIdentifier** value that uniquely identifies the message in the queue.
  - An SRMP message or its internal representation. Storage of the information contained in the SRMP message is implementation-dependent.<10>
  - The time when the message arrived at the queue.
- **MessageIDHistoryTable**: This table contains a history of **MessageIdentifier** values from messages that the protocol host has received. This table provides a lightweight duplicate elimination mechanism. When a message arrives, the **MessageID** value is checked against this table. If the value exists, then the packet MUST be rejected as a duplicate. The length of history this table maintains is implementation-dependent. This value SHOULD<11> be saved to persistent storage.
- **MessageIdOrdinal**: A monotonically increasing value used in **MessageIdentifer**. This value is incremented by 1 for each message the protocol sends. This value MUST be saved to persistent storage.
- **OutgoingMessage table**: An ordered list of **MessageEntry** data elements. This table contains unsent messages and/or messages awaiting receipts. **MessageEntry** elements containing durable or stream messages MUST be saved to persistent storage. This table MUST generate the outgoing message event, as described in section 3.6.1.
- **AwaitingCommitmentReceipt table**: A list of **MessageEntry** data elements. This table contains a list of messages that have been successfully delivered but are awaiting commitment receipt. This table MUST be saved to persistent storage.
- **AckWaitTimeout**: The time, in milliseconds, that the protocol waits before retransmitting a message because its messages are not acknowledged.
- **UnAckedMessageCount**: The count of sent messages requesting delivery receipt that have not received a delivery receipt from the remote QM.
- **MessageEntry**: A data structure containing the following fields:
  - **Message**: An SRMP message.

  Durable: A Boolean value indicating if the message is durable.
- **StreamSequenceNumber**: A stream sequence number. A 0 value indicates that the message is not a stream message.
- **AwaitingDeliveryReceipt**: A Boolean value indicating if the message has been sent and is awaiting delivery receipt.

- **ReceivedDeliveryReceipt**: A Boolean value indicating that the message has received a delivery receipt.
- **ReceivedStreamReceipt**: A Boolean value indicating that the message has received a stream receipt.
- **Transmitted**: A Boolean value indicating that the message has been sent at least once.

3.1.1.3 Stream State

The sender and receiver MUST independently maintain the following data elements for each stream:

- **StreamID**: A stream ID that identifies the current outgoing sequence of stream messages. Only one sequence is valid at a given time. This value consists of a GUID and an ordinal, as described in section 2.2.5.3.1. This value MUST be saved to persistent storage.
- **StreamSequenceNumber**: The sequence number of the last outgoing stream message sent on a stream. This value MUST be saved to persistent storage.
- **IncomingStreamID**: A value that identifies the current incoming stream sequence. This value MUST be saved to persistent storage.
- **IncomingStreamSequenceNumber**: A value that identifies the sequence number of the last stream message received on a stream. This value MUST be saved to persistent storage.

The StreamID, StreamSequenceNumber, IncomingStreamID, and IncomingStreamSequenceNumber state values apply only to stream messages that originate from a host or that are addressed to a final destination queue on a host. Stream messages forwarded through a host are not processed as part of the incoming or outgoing stream sequence.

The preceding conceptual data can be implemented using a variety of techniques. An implementation can implement such data in any way as long as it conforms to the abstract data model.

3.1.1.4 Persistent State Storage

Some protocol data elements MUST be saved in a persistent location that will survive process and node failure. A persistent storage requirement is indicated with a note in the element description that indicates "This value MUST be saved to persistent storage".

3.1.2 Stream Message Sequence

To provide EOIO guarantees for stream messages, SRMP organizes stream messages into sequences. A stream message is identified by a stream identifier, as specified in section 2.2.5.3.1, and a 32-bit stream sequence number. The first message within a stream is assigned the value 1. Only one stream is active at a given time.

SRMP maintains the following stream state:

- **StreamID**: A value that identifies the current outgoing stream. This value consists of a GUID and an ordinal as described in section 2.2.5.3.1.
- **StreamSequenceNumber**: The sequence number of the last stream message sent on a stream.
- **IncomingStreamID**: A value that identifies the last incoming stream.

- **IncomingStreamSequenceNumber**: The sequence number of the last stream message received on a stream.

A stream message contains the <stream> element, as specified in section 2.2.5.3, which indicates the stream ID in the <streamID> element, the sequence number in the <current> element, and the sequence number of the previous message in the <previous> element (if the <previous> element is not present, the previous message has a sequence number of <current> - 1). This information allows the remote host to determine if a message is in order and to identify duplicates.

Because messages can expire, gaps are allowed in stream sequence numbers. The <stream> element includes the previous sequence number so the remote host can determine if the received message follows the prior message that was received.

When all the messages within a stream have been acknowledged, the protocol SHOULD<12> increment the StreamID.Ordinal by 1 and reset the StreamSequenceNumber to 0. This process creates a new stream sequence that MUST be used with subsequent streams. Messages MUST NOT be sent on prior stream sequences.

The receiver utilizes stream sequence numbers when acknowledging receipt of stream messages. Stream ID and sequence number values are specified in the <streamReceipt> element to acknowledge receipt of stream messages.

Stream sequences are end-to-end. Processing of streams MUST be done only by the original sender QM and the final destination QM. An intermediate QM that receives a stream message MUST pass the message to the next destination but perform no processing related to the stream sequence.

3.1.3 Receipts

The SRMP augments the underlying transport with additional levels of acknowledgment that ensure messages are reliably transferred irrespective of failures in transport connection, applications, or nodes.

Message receipts provide a mechanism for the receiver to notify the sender that it has received a message and, optionally, whether it has been saved to disk. When the sender receives a receipt, it can discard the acknowledged messages that it has stored locally.

The sender retransmits unacknowledged messages if it does not receive a receipt within a time-out. The SRMP implements message receipts at both the individual message and transactional sequence layers.

3.1.3.1 Delivery Receipts

The receiver sends a delivery receipt to the sender as a regular SRMP message containing the <deliveryReceipt> element. The purpose of a delivery receipt is to notify the sender that the receiver has received the sender's message and, in the case of a durable message, has stored it to disk. Delivery receipts are sent only if the sender requested them by adding the <deliveryReceiptRequest> element to the message.

The sender SHOULD<13> discard its local copy of a message unless other receipts are still outstanding (that is, stream receipts or commitment receipts).

3.1.3.2 Stream Receipts

The receiver sends a stream receipt to the sender in the form of a regular SRMP message that contains the <streamReceipt> element. The purpose of a stream receipt is to notify the original sender that the final destination has received and successfully stored a stream message to disk.

Stream receipts are always sent for stream messages; there is no SRMP message element for requesting them.

Stream receipts pertain to the stream message sequence.

Stream receipts are end-to-end acknowledgments. Processing of streams MUST be done only by the original sending QM and the final destination QM. An intermediate QM that receives a stream message MUST pass the stream message to the next destination but perform no processing related to the stream sequence.

The <streamReceipt> element contains a stream ID (in the <streamID> element) and a stream sequence number (in the <lastOrdinal> element) that specify the stream message being acknowledged. The receiver MUST acknowledge stream messages in sequence order. For example, if it has received messages 1, 2, and 4 within a sequence it cannot send a receipt for message 4 until it has received, saved to disk, and acknowledged message 3.

A stream receipt MAY<14> acknowledge multiple messages if there are multiple messages to acknowledge when the stream receipt is sent. For example, if the last message acknowledged was 5 and the receiver has received and saved to disk messages 6, 7, and 8, then the receiver MAY set the <lastOrdinal> element to the value 8.

The <lastOrdinal> element specifies to the sender the highest message sequence number that has been received by the receiver and saved to disk. The sender SHOULD<15> discard its local copies of acknowledged stream messages up to the position in the sequence that the sender specified, unless other receipts are still outstanding (that is, commitment receipts).

The receiver MUST send stream receipts to the sender after durably storing a stream message to disk. Stream receipts MAY be delayed for the purpose of batching receipts of multiple stream messages.

3.1.3.3 Commitment Receipts

The receiver sends a commitment receipt to the sender as a regular SRMP message that contains the <commitmentReceipt> element. The purpose of a commitment receipt is to notify the sender that the receiver has received a message from the sender, stored any durable message to disk, inserted the message into its destination queue, and either discarded it or passed it to the receiving application. Commitment receipts are sent only if the sender requested them by adding the <commitmentReceiptRequest> element to the message, as specified in section 2.2.5.2.3.

If the sender has specified the <positiveOnly/> element, the receiver MUST send a positive commitment receipt if the receiving application has successfully retrieved the message from the destination queue. If the sender has specified the <negativeOnly/> element, the receiver MUST send a negative commitment receipt if the message is discarded from the destination queue because it expired or because the queue was purged.

If the sender specifies neither <positiveOnly/> nor <negativeOnly/>, the receiver MUST NOT send a commitment receipt.

The sender SHOULD<16> discard its local copy of a message after receiving a commitment receipt.

3.1.4 Sequence Diagrams

This section contains sequence diagrams that illustrate several common scenarios.

3.1.4.1 Regular SRMP Message and Receipts

The following figure shows the message sequence for sending a regular or durable SRMP message between two QMs. In this case, the sender is requesting delivery and commitment receipts from the receiver.

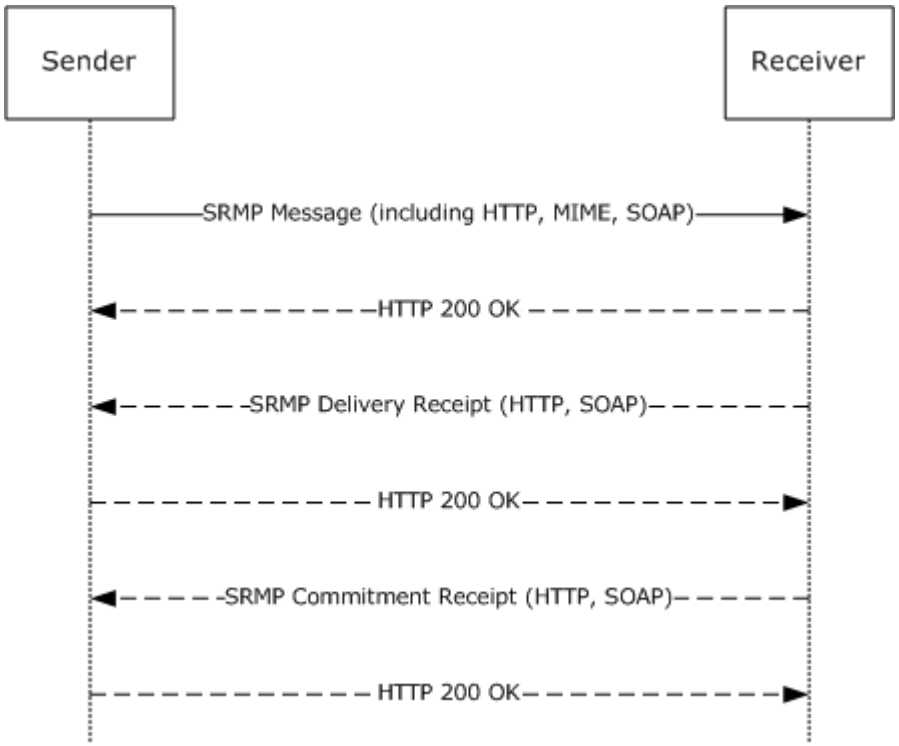

Figure 7: Regular SRMP message and receipts

First, the sender sends the SRMP message to the remote host via HTTP transport. The application message payload is included as a MIME attachment, as specified in section 2.2.2. The HTTP transport acknowledges every successful message by replying with "HTTP 200 OK".

Next, the receiver acknowledges receipt of the message by sending a delivery receipt. The commitment receipt is sent after a delay, when the message is either discarded or picked up by the receiving application.

3.1.4.2 Stream Sequence and Receipts

The following figure illustrates the sending of a message stream between two QMs. In this case, the sender is not requesting delivery or commitment receipts for the individual messages.

Initially, the sender sends a stream message with sequence number 1 to the remote host. As before, the HTTP transport acknowledges every successful message by replying with "HTTP 200 OK".

Next, the sender sends a stream message with sequence number 2 to the remote host. The remote host responds by sending a stream receipt message. The purpose of the stream receipt is to acknowledge that the stream message was received in the correct order and was not a duplicate. By setting the <lastOrdinal> element in the stream receipt to 2, the receiver acknowledges receipt of all messages up to sequence number 2. A separate receipt for stream message 1 is not necessary.

Finally, the sender sends another stream message with sequence number 3, and the remote host acknowledges it with a stream receipt with the <lastOrdinal> set to 3.

Since SRMP uses HTTP 1.1 as a transport, which is a connectionless protocol, no separate messages need to be exchanged for establishing or closing a connection. The connectionless nature of SRMP is in contrast to the connection-oriented Message Queuing Binary Protocol ([MS-MQQB]), which otherwise serves the same purpose.

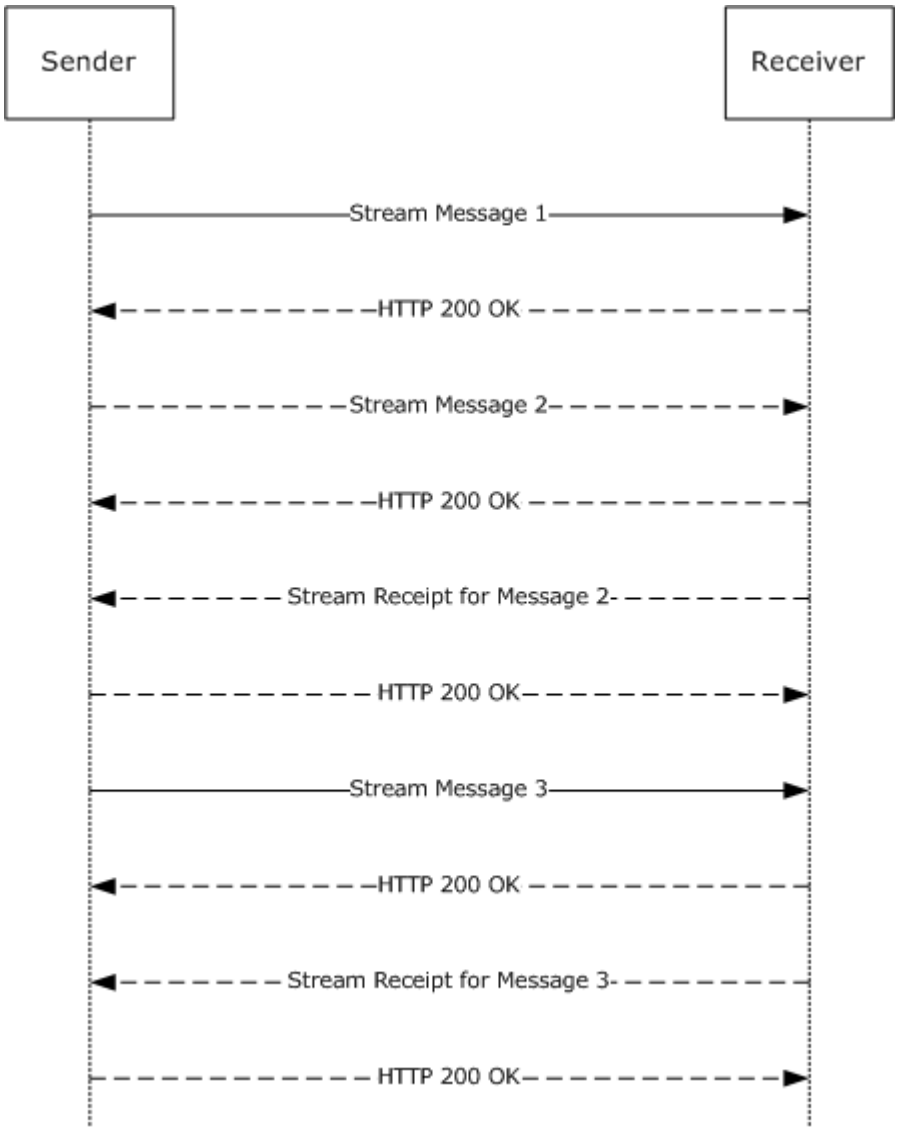

Figure 8: Stream sequence and receipts

3.1.4.3 Stream Message and Multiple Receipts

The following diagram illustrates the sending of an individual stream message between two QMs. In this case, the sender is requesting delivery and commitment receipts in addition to the mandatory stream receipt. The mandatory "HTTP 200 OK" messages that the HTTP transport sends in response to each message in this diagram have been omitted.

First, the sender sends a stream message to the remote host. The receiver responds by sending a delivery receipt message. Next, the receiver sends the mandatory stream receipt message, acknowledging receipt of all stream messages up to the current message. It is important to note that delivery receipts and stream receipts are separate mechanisms that serve different purposes.

Finally, the remote host sends a commitment receipt to the sender when the receiving application consumes the message from the destination queue, or when it is discarded from the destination queue. Note that the diagram shows only the sequence for a single stream message — there can be many stream receipts acknowledging multiple messages before the commitment receipt for this message is sent.

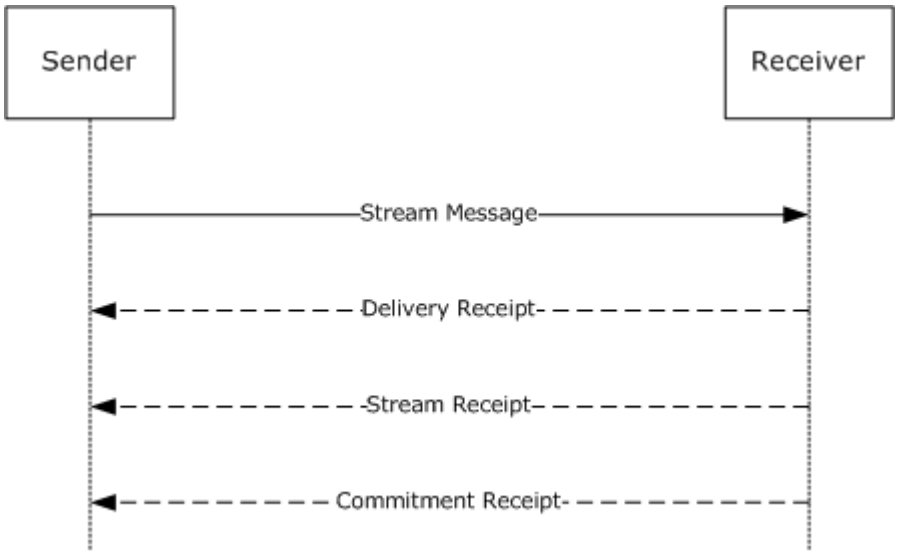

Figure 9: Stream message and multiple receipts

3.2 Timers

The protocol MUST maintain the following timers.

3.2.1 Delivery Receipt Wait Timer

The Delivery Receipt Wait Timer regulates the amount of time that the protocol waits for a delivery receipt before retransmitting the unacknowledged message. This timer is armed after sending a message requesting a delivery receipt. The duration of this timer MUST be set to AckWaitTimeout.

3.2.2 Stream Receipt Wait Timer

The Stream Receipt Wait Timer regulates the amount of time that the protocol waits for a stream receipt message before resending stream messages to the remote host. This timer is started after sending a stream message. The duration of this timer SHOULD be set based on system configuration, which is implementation-dependent.<17>

3.3 Initialization

3.3.1 Global Initialization

The following values MUST be initialized globally:

- The MessageIDHistoryTable MUST be loaded from persistent storage. If the value does not exist in persistent storage, then it MUST be set to an empty table.
- The Delivery Receipt Wait Timer MUST be disabled.
- The OutgoingMessageTable MUST be loaded from persistent storage if it does not exist in memory from a previous protocol instance. If the table does not exist in memory or persistent storage, then it MUST be initialized to an empty table. The **AwaitingDeliveryReceipt**, **ReceivedDeliveryReceipt**, **ReceivedStreamReceipt**, and **Transmitted** fields of each MessageEntry table entry MUST be set to FALSE.
- The value of MessageIdOrdinal MUST be loaded from persistent storage. If the table does not exist in persistent storage, then it MUST be initialized to the value 0x00000000.
- The value of AckWaitTimeout MUST be set based on system configuration, which is implementation-dependent.<18>
- The value of AckSendTimeout MUST be set based on system configuration, which is implementation-dependent.
- The value of UnAckedMessageCount MUST be set to the count of entries in the OutgoingMessage table where ReceivedDeliveryReceipt is false AND AwaitingDeliveryReceipt is true.
- The value of QMGuid MUST be globally unique and set based on system configuration, which is implementation-dependent.<19>

3.3.2 Stream Initialization

The following values MUST be initialized for each stream:

- The value of StreamID MUST be loaded from persistent storage. If the value does not exist in persistent storage, then StreamID.Ordinal MUST be set to 0x00000001, StreamID.GUID MUST be set to an implementation-dependent value that is guaranteed to be different from any previously generated value, and StreamSequenceNumber MUST be set to 0x00000000.
- The value StreamSequenceNumber MUST be loaded from persistent storage. If the value does not exist in persistent storage, then it MUST be set to 0.
- The Stream Receipt Wait Timer MUST be disabled.
- The value of IncomingStreamID MUST be loaded from persistent storage. If the value does not exist in persistent storage, then it MUST be set to 0x0000000000000000.
- The value of IncomingStreamSequenceNumber MUST be loaded from persistent storage. If the value does not exist in persistent storage, then it MUST be set to 0x00000000.

3.4 Higher-Layer Triggered Events

The operation of the SRMP is initiated and subsequently driven by the following higher-layer triggered events:

- The QM service starting.
- The QM service stopping.
- The QM inserting a message into the OutgoingMessageTable.

- The administrator purging a queue.

3.4.1 Queue Manager Service Started

On startup, the QM service MUST restore the protocol persistent state, as described in section 3.3.

3.4.2 Queue Manager Service Stopped

No specific processing is required when the QM service is stopped.

3.4.3 Queue Manager Inserts Message into OutgoingMessage Table

This event occurs when the QM inserts a MessageEntry into the OutgoingMessage table. The MessageEntry MUST have the following values:

- The **Message** field is set to an SRMP message.
- The **AwaitingDeliveryReceipt**, **ReceivedDeliveryReceipt**, **ReceivedStreamReceipt**, and **Transmitted** fields are set to FALSE.
- The **StreamSequenceNumber** field is set to 0x00000000.

For information on how the message is processed, see section 3.6.1.

3.4.4 Administrator Purges a Queue

This event occurs when the administrator purges the messages from a queue hosted by the QM. All messages MUST be removed from a purged queue. Each removed message MUST be processed as described in section 3.6.2.

3.5 Message Processing Events and Sequencing Rules

3.5.1 Receiving Any Message

Unless specifically noted in a subsequent section, the following logic MUST be applied to any message received.

3.5.1.1 Identifying the Message Type

A message is identified by inspecting the contents of its SOAP <header> element (see section 2.2.2). The following section describes how to identify each message type. For details on message class identifiers, see [MS-MQQB] section **2.2.1.5**.

Delivery Receipt Message: The <deliveryReceipt> element MUST be present. The <Class> element of the <Msmq> element MUST NOT be set to MQMSG_CLASS_NORMAL.

Stream Receipt Message: The <streamReceipt> element MUST be present. The <action> element of the <path> element MUST be set to "MSMQ:QM Ordering Ack". The <Class> element of the <Msmq> element MUST NOT be set to MQMSG_CLASS_NORMAL.

Commitment Receipt Message: The <commitmentReceipt> element MUST be present. The <Class> element of the <Msmq> element MUST NOT be set to MQMSG_CLASS_NORMAL.

User Message: The <Class> element of the <Msmq> element MUST be set to MQMSG_CLASS_NORMAL.

3.5.1.2 Handling Incorrectly Formatted Messages

If the protocol receives a request that does not conform to the structures outlined in section 2, the protocol MUST discard the received packet and perform no further processing for it.

3.5.2 Receiving a Delivery Receipt Message

The protocol MUST perform the following steps to process a delivery receipt message.

3.5.2.1 Marking an Acknowledged Message

The protocol MUST set the **MessageEntry.ReceivedDeliveryReceipt** field to TRUE for the corresponding entry in the OutgoingMessage table where the <id> element in the <path> element equals the <id> element in the <deliveryReceipt> element.

UnAckedMessageCount MUST be decremented by 1.

3.5.2.2 Deleting an Acknowledged Message

The protocol MUST delete all elements from the OutgoingMessage table where **MessageEntry.ReceivedDeliveryReceipt** is set to TRUE and **MessageEntry.StreamSequenceNumber** is set to 0x00000000 (indicating that this is not a stream message).

The protocol MUST delete all elements from the OutgoingMessage table where **MessageEntry.ReceivedDeliveryReceipt** is set to TRUE and **MessageEntry.ReceivedStreamReceipt** is set to TRUE (indicating that this is a stream message that has already received a stream receipt).

Stream messages MUST be retained in the OutgoingMessage table until receipt of the corresponding stream receipt message.

A message that is deleted from the OutgoingMessage table with the <commitmentReceiptRequest> element present MUST be copied to the AwaitingCommitmentReceipt table.

If, after deleting acknowledged messages, the OutgoingMessage table contains any message where **AwaitingDeliveryReceipt** is TRUE (that is, UnAckedMessageCount is greater than 0) then the Delivery Receipt Wait timer MUST be restarted.

3.5.2.3 Source Journaling

An acknowledged message that is deleted from the OutgoingMessage table with the <Journal/> element present in the <Msmq> element SHOULD<20> be logged locally.

Source journaling is deferred for regular and durable messages that are copied to the AwaitingCommitmentReceipt table until the commitment receipt is received.

3.5.3 Receiving a Stream Receipt Message

The protocol MUST perform the following steps to process a stream receipt message.

3.5.3.1 Marking an Acknowledged Message

The SRMP MUST set the **MessageEntry.ReceivedStreamReceipt** field to TRUE for all entries in the OutgoingMessageTable that have the following:

- The <streamID> element in the <stream> element equals the <streamID> element in the <streamReceipt> element.
- The **MessageEntry.StreamSequenceNumber** is greater than 0x00000000 and is less than or equal to the <lastOrdinal> element in the <streamReceipt> element.

3.5.3.2 Deleting an Acknowledged Message

The protocol MUST delete all elements from the OutgoingMessage table where **MessageEntry.StreamSequenceNumber** is greater than 0x00000000, **MessageEntry.ReceivedStreamReceipt** is set to TRUE, and **MessageEntry.ReceivedDeliveryReceipt** is set to TRUE.

The protocol MUST delete all elements from the OutgoingMessage table where **MessageEntry.StreamSequenceNumber** is greater than 0x00000000, **MessageEntry.ReceivedStreamReceipt** is set to TRUE, and **MessageEntry.AwaitingDeliveryReceipt** is set to FALSE.

A message that is deleted from the OutgoingMessage table with the <commitmentReceiptRequest> element present MUST be copied to the AwaitingCommitmentReceipt table.

If the OutgoingMessage table contains no MessageEntry elements where **MessageEntry.StreamSequenceNumber** is nonzero (that is, there are no stream receipts outstanding) then the protocol SHOULD<21> increment **StreamID.Ordinal** by 1 and set **StreamSequenceNumber** to the value 0x00000000.

3.5.3.3 Source Journaling

A stream message that is deleted from the OutgoingMessage table with the <Journal/> element present in the <Msmq> element SHOULD<22> be logged locally.

Source journaling is deferred for stream messages that are copied to the AwaitingCommitmentReceipt table until the commitment receipt is received.

3.5.4 Receiving a Commitment Receipt Message

A commitment receipt message indicates that the message represented by the <id> element in the <commitmentReceipt> element has been removed from the destination queue. The protocol MUST perform the following steps to process a commitment receipt message:

1. The protocol MUST delete the message from the AwaitingCommitmentReceipt table where the <id> element in the <path> element is equal to the <id> element in the commitment receipt element.
2. If the value of the <decision> element in the commitment receipt element is "positive", the message MUST be moved to the system queue journal.
3. If the value of the <decision> element in the commitment receipt element is "negative", the message MUST be moved to the system transactional dead-letter queue.

The journal and dead-letter queues are system-generated and implementation-dependent.<23>

3.5.5 Receiving a User Message

A user message contains an application-defined message sent from the remote host. A received message can be addressed to a local queue or to a queue on a remote host.

Processing a user message consists of the following sequence of operations:

- Detecting duplicates
- General processing
- Checking message expiration
- Processing stream messages
- Processing regular and durable messages
- Inserting a message into a local queue

The protocol MUST perform all these steps, described in greater detail in the following sections, in order to process a user message.

3.5.5.1 Detecting Duplicates

If the value in the <id> element of the <path> element exists in the MessageIDHistoryTable table, the protocol MUST discard this message and perform no further processing.

3.5.5.2 General Processing

A MessageIdentifier consisting of the value in the <id> element of the <path> element<24> MUST be inserted into the MessageIDHistoryTable.

The protocol MUST perform the following error handling logic:

- The protocol MUST disregard a message if it is addressed to a nonexistent queue. If the <to> element of the <path> element does not correspond to a queue in QueueTable, then the protocol MUST disregard the message and perform no further processing.
- A stream message can be delivered only to a transactional queue. If the message contains the <stream> element and is addressed to a nontransactional queue in QueueTable, then the protocol MUST disregard the message and perform no further processing.
- A nonstream message can be delivered only to a nontransactional queue. If the message does not contain the <stream> element and is addressed to a transactional queue in QueueTable, then the protocol MUST disregard the message and perform no further processing.
- If the packet is rejected for any reason other than being a duplicate, then the protocol MUST return the "HTTP 400 Bad Request" error message to the sender containing the string: "Bad SRMP".

3.5.5.3 Checking Message Expiration

A message can contain a nonzero <TTrq> element in the <Msmq> element and/or the <expiresAt> / <duration> elements in the <properties> element value that controls message lifetime. The protocol MUST check a received message for expiration before processing.

CURRENT_TIME represents the current system time. This value is the number of seconds elapsed since midnight (00:00:00), January 1, 1970 UTC, according to the system clock.

- If CURRENT_TIME - <sentAt> is greater than <TTrq>, then the message has expired. The protocol MUST disregard the message and perform no further processing.

- If CURRENT_TIME - <sentAt> is greater than <duration>, then the message has expired. The protocol MUST disregard the message and perform no further processing.
- If CURRENT_TIME is greater than <expiresAt>, then the message has expired. The protocol MUST disregard the message and perform no further processing.

If a message that has expired contains a <DeadLetter/> value in the <Msmq> element, then the message SHOULD<25> be logged locally.

3.5.5.4 Processing Stream Messages

Stream messages are accepted only in order and exactly once. If the message has not been rejected for any reason previously stated, the message MUST be accepted if one of the following is true:

- The <stream> element contains the <start> element, and its <current> element is set to 1. In this case, the protocol MUST set IncomingStreamID to the content of the message's <streamID> element.

  OR

- The <streamID> element is equal to IncomingStreamID, and its <current> element is equal to IncomingStreamSequenceNumber + 1.

  OR

- The <streamID> element is equal to IncomingStreamID, its <current> element is greater than IncomingStreamSequenceNumber, and its <previous> element is less than or equal to IncomingStreamSequenceNumber.

If the message is accepted, the protocol MUST set IncomingStreamSequenceNumber to the message's <current> value.

A stream receipt message MUST be sent to the remote host to acknowledge the message. The protocol MAY<26> delay sending the stream receipt to provide batching of receipts for multiple stream messages. The <streamID> element in the <streamReceipt> element MUST be set to IncomingStreamID, and the <lastOrdinal> element MUST be set to IncomingStreamSequenceNumber.

The stream receipt message MUST be addressed to the sender order queue by setting the <to> element of the <path> element to the URI in the <sendReceiptsTo> element of the <start> element of the first stream message.

3.5.5.5 Processing Regular and Durable Messages

If the <durable/> element in the <properties> element is present, or if the <stream> element is present, the protocol MUST save the message to disk.

If the <deliveryReceiptRequest> element is present in the message, then the protocol MUST send a delivery receipt to acknowledge that the message has been received and durably stored.

3.5.5.6 Inserting a Message into a Local Queue

If the URI value contained in the <to> element of the <path> element resolves to a queue on the local machine, then the message MUST be inserted into the corresponding queue in QueueTable.

If the URI value contained in the <to> element of the <path> element does not resolve to a queue on the local machine, then the message MUST be inserted into an outgoing queue where it will be processed by the QM and routed to a host closer to the destination.

3.5.5.7 Timer Events

3.5.5.7.1 Delivery Receipt Wait Timer Event

This event indicates a time-out while waiting for a delivery receipt from the remote host. When the Delivery Receipt Wait Timer fires, the protocol MUST retransmit all messages in the OutgoingMessage table that have **MessageEntry.AwaitingDeliveryReceipt** set to TRUE.

3.5.5.7.2 Stream Receipt Wait Timer Event

This event indicates a time-out while waiting for a stream receipt from the remote host. When the Stream Receipt Wait Timer fires, the protocol MUST retransmit all messages in the OutgoingMessage table that have **MessageEntry.StreamSequenceNumber** greater than 0x00000000 and **MessageEntry.ReceivedStreamReceipt** set to FALSE.

3.6 Other Local Events

3.6.1 Outgoing Message Event

This event indicates that there is a MessageEntry in the OutgoingMessageTable ready to be sent to the remote QM. The event provides a reference to the corresponding MessageEntry.

This event is triggered when a MessageEntry is inserted into the OutgoingMessageTable with the **MessageEntry.Transmitted** field set to FALSE.

The following steps MUST be performed to send the message:

1. Check for message expiration.
2. Update the SRMP message elements.
3. Send the message.

Unless specifically noted in a subsequent section, the following logic MUST be applied to any user message sent.

3.6.1.1 Checking for Message Expiration

For information on message expiration, see section 3.5.5.3.

3.6.1.2 Updating the SRMP Message Elements

The protocol MUST set the <id> element in the <path> element to a string representing the current MessageIdOrdinal (the message index) and the source QM GUID, as described in section 2.2.4.3. The value of **MessageIdOrdinal** MUST be incremented by 1.

If the message is a stream message, then the following steps MUST be performed:

1. The <streamID> element in the <stream> element MUST be set to StreamID.
2. The StreamSequenceNumber value MUST be incremented by 1.

3. The <current> element in the <stream> element MUST be set to StreamSequenceNumber.

4. If the previous stream message expired, the <previous> element in the <stream> element MUST be set to the StreamSequenceNumber of the previous transactional message in the OutgoingMessage table in order to bridge the gap in sequence numbers.

5. **MessageEntry.StreamSequenceNumber** MUST be set to StreamSequenceNumber.

6. The Stream Receipt Wait timer MUST be started.

The value of UnAckedMessageCount MUST be incremented by 1.

If <deliveryReceiptRequest> is present, the value of **MessageEntry.AwaitingDeliveryReceipt** MUST be set to TRUE.

The protocol MUST start the Delivery Receipt Wait timer if it is in the stopped state.

3.6.1.3 Sending the Packet

The user message MUST be sent to the remote QM using the HTTP 1.1 transport.

The protocol MUST set **MessageEntry.Transmitted** to TRUE.

3.6.2 Message Removed from Destination Queue

A message can be removed from a destination queue because the message was read by a higher-layer application, the message expired, or the queue was deleted or purged, as described in section 3.4.4. Operations that occur on messages in a destination queue are outside the definition of this protocol; however, the protocol must ensure that messages are tracked and that the following acknowledgement logic is applied.

3.6.2.1 Commitment Receipt

If the <commitmentReceiptRequest> element is present in a user message, the protocol MUST send a commitment receipt message to the original sender when the message is removed from the destination queue. If the <positiveOnly/> element is present, a positive commitment receipt MUST be sent if the message is received by the receiving application. If the <negativeOnly/> element is present, a negative commitment receipt MUST be sent if the message expires or is purged from the destination queue before reaching the receiving application. If both <positiveOnly/> and <negativeOnly/> elements are present, both types of commitment receipts MUST be sent; if neither element is present, commitment receipts MUST NOT be sent.

The <id> element in the <commitmentReceipt> element MUST be set to the <id> element of the <path> element of the original message requesting the receipt.

The <decidedAt> element must be set to the CURRENT_TIME at which the outcome was recorded, and the <decision> element MUST be set to "positive" for a positive receipt or to "negative" for a negative receipt.

The commitment receipt message MUST be addressed by setting the <to> element of the <path> element to the URI in the <sendTo> element of the <commitmentReceiptRequest> element.

3.6.3 Handling a Network Disconnect

Since the SRMP is connectionless, no specific steps are required for handling a network disconnect.

4 Protocol Examples

4.1 Simple SRMP Message

This sample message contains the minimum fields necessary to send a message without any receipts expected. This example also includes the HTTP and MIME fields necessary to send the message.

```
POST /msmq/private$/simpleq HTTP/1.1
Host: machine2
Content-Type: multipart/related; boundary="MSMQ - SOAP boundary, 53287";
type=text/xml
Content-Length: 906
SOAPAction: "MSMQMessage"
Proxy-Accept: NonInteractiveClient

--MSMQ - SOAP boundary, 53287
Content-Type: text/xml; charset=UTF-8
Content-Length: 619

<se:Envelope xmlns:se="http://schemas.xmlsoap.org/soap/envelope/"
xmlns="http://schemas.xmlsoap.org/srmp/">
  <se:Header>
    <path xmlns="http://schemas.xmlsoap.org/rp/" se:mustUnderstand="1">
      <action>MSMQ:mqsender label</action>
      <to>http://machine2/msmq/private$/simpleq</to>
      <id>uuid:1@00000000-0000-0000-0000-000000000000</id>
    </path>
    <properties se:mustUnderstand="1">
      <expiresAt>20070609T164419</expiresAt>
      <sentAt>20070608T164419</sentAt>
    </properties>
  </se:Header>
  <se:Body></se:Body>
</se:Envelope>--MSMQ - SOAP boundary, 53287
Content-Type: application/octet-stream
Content-Length: 13
Content-Id: body@ff3af301-3196-497a-a918-72147c871a13

First Message--MSMQ - SOAP boundary, 53287--
```

4.2 Simple Message Including MSMQ Element

The following is a simple message that includes the MSMQ element and a slightly more complex message payload.

```
POST /msmq/private$/simpleq HTTP/1.1
Host: machine2
Content-Type: multipart/related; boundary="MSMQ - SOAP boundary, 26500";
type=text/xml
Content-Length: 1273
SOAPAction: "MSMQMessage"
Proxy-Accept: NonInteractiveClient

--MSMQ - SOAP boundary, 26500
Content-Type: text/xml; charset=UTF-8
Content-Length: 775

<se:Envelope xmlns:se="http://schemas.xmlsoap.org/soap/envelope/"
```

```
xmlns="http://schemas.xmlsoap.org/srmp/">
  <se:Header>
    <path xmlns="http://schemas.xmlsoap.org/rp/" se:mustUnderstand="1">
      <action>MSMQ:</action>
      <to>http://machine2/msmq/private$/simpleQ</to>
      <id>uuid:20503@caf195ea-615c-4264-ae08-11a4e60194c0</id>
    </path>
    <properties se:mustUnderstand="1">
      <expiresAt>20380119T031407</expiresAt>
      <sentAt>20070719T031140</sentAt>
    </properties>
    <Msmq xmlns="msmq.namespace.xml">
      <Class>0</Class>
      <Priority>3</Priority>
      <Correlation>AAAAAAAAAAAAAAAAAAAAAAAAAAA=</Correlation>
      <App>0</App>
      <BodyType>0</BodyType>
      <HashAlgorithm>32772</HashAlgorithm>
      <SourceQmGuid>caf195ea-615c-4264-ae08-11a4e60194c0</SourceQmGuid>
      <TTrq>20070723T031140</TTrq>
    </Msmq>
  </se:Header>
<se:Body></se:Body>
</se:Envelope>--MSMQ - SOAP boundary, 26500
Content-Type: application/octet-stream
Content-Length: 223
Content-Id: body@caf195ea-615c-4264-ae08-11a4e60194c0

<?xml version="1.0"?>
<Order xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <orderId>3</orderId>
  <orderTime>2007-07-18T20:11:40.2614595-07:00</orderTime>
</Order>--MSMQ - SOAP boundary, 26500--
```

4.3 Combined Delivery and Commitment Receipt Request Example

The following example requests both a delivery receipt and a commitment receipt acknowledgment in the same message.

```
POST /msmq/private$/simpleq HTTP/1.1
Host: machine2
Content-Type: multipart/related; boundary="MSMQ - SOAP boundary, 95692";
type=text/xml
Content-Length: 1553
SOAPAction: "MSMQMessage"
Proxy-Accept: NonInteractiveClient
--MSMQ - SOAP boundary, 95692
Content-Type: text/xml; charset=UTF-8
Content-Length: 1233

<se:Envelope xmlns:se= "http://schemas.xmlsoap.org/soap/envelope/"
             xmlns:rp="http://schemas.xmlsoap.org/rp/"
             xmlns="http://schemas.xmlsoap.org/srmp/">
  <se:Header>
    <rp:path se:mustUnderstand="1">
      <rp:action>Generic label</rp:action>
      <rp:to>http://machine2/msmq/private$/simpleq</rp:to>
      <rp:id>uuid:1@00000000-0000-0000-0000-000000000008</rp:id>
      <rp:rev>
        <rp:via>http://machine1/MSMQ/private$/Q1</rp:via>
      </rp:rev>
```

```
    </rp:path>
    <properties se:mustUnderstand="1">
      <expiresAt>20070720T032452</expiresAt>
      <sentAt>20070719T032452</sentAt>
    </properties>
    <services xmlns="http://schemas.xmlsoap.org/srmp/"
    se:mustUnderstand="1">
      <commitmentReceiptRequest>
        <sendTo>http://machine1/MSMQ/private$/deliverydone</sendTo>
        <negativeOnly/>
        <positiveOnly/>
      </commitmentReceiptRequest>
      <deliveryReceiptRequest>
        <sendTo>http://machine1/MSMQ/private$/receipts</sendTo>
      </deliveryReceiptRequest>
    </services>
  </se:Header>
  <se:Body></se:Body>
</se:Envelope>--MSMQ - SOAP boundary, 95692
Content-Type: application/octet-stream
Content-Length: 45
Content-Id: body@ff3af301-3196-497a-a918-72147c871a13
```

Both delivery and commitment receipt requests—MSMQ - SOAP boundary, 95692—

The following is the HTTP 1.1 response to the initial message.

```
HTTP/1.1 200 OK
Content-Length: 0
Server: Microsoft-IIS/7.0
Date: Thu, 19 Jul 2007 03:24:51 GMT
```

The following is the delivery receipt.

```
POST /msmq/private$/receipts HTTP/1.1
Host: machine1
Content-Type: text/xml
Content-Length: 959
SOAPAction: "MSMQMessage"
Proxy-Accept: NonInteractiveClient

<se:Envelope xmlns:se="http://schemas.xmlsoap.org/soap/envelope/"
xmlns="http://schemas.xmlsoap.org/srmp/">
  <se:Header>
    <path xmlns="http://schemas.xmlsoap.org/rp/" se:mustUnderstand="1">
      <action>MSMQ:</action>
      <to>http://machine1/MSMQ/private$/receipts</to>
      <rev>
        <via>http://machine2/msmq/private$/simpleq</via>
      </rev>
      <id>uuid:34826@ac678228-2dd6-418b-b31f-0539ffeea853</id>
    </path>
    <properties se:mustUnderstand="1">
      <expiresAt>20380119T031407</expiresAt>
      <sentAt>20070719T032451</sentAt>
    </properties>
```

```
    <deliveryReceipt>
      <receivedAt>20070719T032454</receivedAt>
      <id>uuid:1@00000000-0000-0000-0000-000000000000</id>
    </deliveryReceipt>
    <Msmq xmlns="msmq.namespace.xml">
      <Class>2</Class>
      <Priority>3</Priority>
      <Correlation>AAAAAAAAAAAAAAAAAAAAAAEAAAA=</Correlation>
      <App>0</App>
      <BodyType>0</BodyType>
      <HashAlgorithm>0</HashAlgorithm>
      <SourceQmGuid>ac678228-2dd6-418b-b31f-0539ffeea853</SourceQmGuid>
      <TTrq>20070723T032451</TTrq>
    </Msmq>
  </se:Header>
  <se:Body></se:Body>
</se:Envelope>
```

The following is the HTTP 1.1 response to the delivery receipt.

```
HTTP/1.1 200 OK
Content-Length: 0
Server: Microsoft-IIS/7.0
Date: Thu, 19 Jul 2007 03:24:57 GMT
```

The following is the commitment receipt.

```
POST /msmq/private$/receipts HTTP/1.1
Host: machine1
Content-Type: text/xml
Content-Length: 994
SOAPAction: "MSMQMessage"
Proxy-Accept: NonInteractiveClient

<se:Envelope xmlns:se="http://schemas.xmlsoap.org/soap/envelope/"
xmlns="http://schemas.xmlsoap.org/srmp/">
  <se:Header>
    <path xmlns="http://schemas.xmlsoap.org/rp/" se:mustUnderstand="1">
      <action>MSMQ:</action>
      <to>http://machine1/MSMQ/private$/receipts</to>
      <rev>
        <via>http://machine2/msmq/private$/simpleq</via>
      </rev>
      <id>uuid:34827@ac678228-2dd6-418b-b31f-0539ffeea853</id>
    </path>
    <properties se:mustUnderstand="1">
      <expiresAt>20380119T031407</expiresAt>
      <sentAt>20070719T032721</sentAt>
    </properties>
    <commitmentReceipt>
      <decidedAt>20070719T032721</decidedAt>
      <decision>positive</decision>
      <id>uuid:1@00000000-0000-0000-0000-000000000000</id>
    </commitmentReceipt>
```

```
    <Msmq xmlns="msmq.namespace.xml">
      <Class>16384</Class>
      <Priority>3</Priority>
      <Correlation>AAAAAAAAAAAAAAAAAAAAAAEAAAA=</Correlation>
      <App>0</App>
      <BodyType>0</BodyType>
      <HashAlgorithm>0</HashAlgorithm>
      <SourceQmGuid>ac678228-2dd6-418b-b31f-0539ffeea853</SourceQmGuid>
      <TTrq>20070723T032721</TTrq>
    </Msmq>
  </se:Header>
  <se:Body></se:Body>
</se:Envelope>
```

The following is the HTTP 1.1 response to the commitment receipt.

```
HTTP/1.1 200 OK
Content-Length: 0
Server: Microsoft-IIS/7.0
Date: Thu, 19 Jul 2007 03:27:21 GMT
```

4.4 Stream Sample

The following is an example of the messages exchanged in a stream. For the sake of brevity, the HTTP 1.1 200 acknowledgements are left out.

```
POST /msmq/private$/tsimpleq HTTP/1.1
Host: machine2
Content-Type: multipart/related; boundary="MSMQ - SOAP boundary, 1672";
type=text/xml
Content-Length: 1750
SOAPAction: "MSMQMessage"
Proxy-Accept: NonInteractiveClient

--MSMQ - SOAP boundary, 1672
Content-Type: text/xml; charset=UTF-8
Content-Length: 1465

<se:Envelope xmlns:se="http://schemas.xmlsoap.org/soap/envelope/"
xmlns="http://schemas.xmlsoap.org/srmp/">
  <se:Header>
    <path xmlns="http://schemas.xmlsoap.org/rp/" se:mustUnderstand="1">
      <action>MSMQ:mqsender label</action>
      <to>http://machine2/msmq/private$/tsimpleq</to>
      <id>uuid:1@00000000-0000-0000-0000-000000000008</id>
    </path>
    <properties se:mustUnderstand="1">
      <expiresAt>20070620T165959</expiresAt>
      <sentAt>20070619T165959</sentAt>
    </properties>
    <Stream se:mustUnderstand="1">
      <streamId>uid:2744e4e1-2b48-43e8-b441-42745f280d53\111</streamId>
      <current>1</current>
      <start>
        <sendReceiptsTo>
          http://machine1/MSMQ/private$/receipts<b>SenderStream=XRntV</b>
        </sendReceiptsTo>
```

```
        <expiresAt>20070620T165959</expiresAt>
      </start>
      <streamReceiptRequest/>
    </Stream>
    <Msmq xmlns="msmq.namespace.xml">
      <Class>0</Class>
      <Priority>0</Priority>
      <Correlation>AAAAAAAAAAAAAAAAAAAAAAAAAAA=</Correlation>
      <App>0</App>
      <BodyType>0</BodyType>
      <HashAlgorithm>32772</HashAlgorithm>
      <SourceQmGuid>dc1cd9a6-8130-4504-88d2-851707fe4632</SourceQmGuid>
      <TTrq>20070620T165959</TTrq>
    </Msmq>
  </se:Header>
  <se:Body></se:Body>
</se:Envelope>--MSMQ - SOAP boundary, 1672
Content-Type: application/octet-stream
Content-Length: 13
Content-Id: body@ff3af301-3196-497a-a918-72147c871a13

First Message--MSMQ - SOAP boundary, 1672--
```

Stream receipt message (ordering ack) acknowledging the first message:

```
POST /msmq/private$/receipts?senderstream=xrntv HTTP/1.1
Host: machine1
Content-Type: text/xml
Content-Length: 993
SOAPAction: "MSMQMessage"
Proxy-Accept: NonInteractiveClient

<se:Envelope xmlns:se="http://schemas.xmlsoap.org/soap/envelope/"
xmlns="http://schemas.xmlsoap.org/srmp/">
  <se:Header>
    <path xmlns="http://schemas.xmlsoap.org/rp/" se:mustUnderstand="1">
      <action>MSMQ:QM Ordering Ack</action>
      <to>http://machine1/MSMQ/private$/receipts?SenderStream=XRntV</to>
      <rev>
        <via>http://machine2/msmq/private$/tsimpleq</via>
      </rev>
      <id>uuid:26641@32221eda-9376-46df-b6ed-783091123831</id>
    </path>
    <properties se:mustUnderstand="1">
      <expiresAt>20380119T031407</expiresAt>
      <sentAt>20070619T165958</sentAt>
    </properties>
    <streamReceipt>
      <streamId>uid:2744e4e1-2b48-43e8-b441-42745f280d53\111</streamId>
      <lastOrdinal>1</lastOrdinal>
    </streamReceipt>
    <Msmq xmlns="msmq.namespace.xml">
      <Class>255</Class>
      <Priority>0</Priority>
      <Correlation>AAAAAAAAAAAAAAAAAAAAAAAAAAA=</Correlation>
      <App>0</App>
      <BodyType>0</BodyType>
      <HashAlgorithm>0</HashAlgorithm>
      <SourceQmGuid>32221eda-9376-46df-b6ed-783091123831</SourceQmGuid>
      <TTrq>20070623T165958</TTrq>
    </Msmq>
  </se:Header>
```

```
  <se:Body></se:Body>
</se:Envelope>
```

Second message in stream:

```
POST /msmq/private$/tsimpleq HTTP/1.1
Host: machine2
Content-Type: multipart/related; boundary="MSMQ - SOAP boundary, 1672"; type=text/xml
Content-Length: 1500
SOAPAction: "MSMQMessage"
Proxy-Accept: NonInteractiveClient

--MSMQ - SOAP boundary, 1672
Content-Type: text/xml; charset=UTF-8
Content-Length: 1220

<se:Envelope xmlns:se="http://schemas.xmlsoap.org/soap/envelope/"
xmlns="http://schemas.xmlsoap.org/srmp/">
  <se:Header>
    <path xmlns="http://schemas.xmlsoap.org/rp/" se:mustUnderstand="1">
      <action>MSMQ:mqsender label</action>
      <to>http://machine2/msmq/private$/tsimpleq</to>
      <id>uuid:1@00000000-0000-0000-0000-000000000008</id>
    </path>
    <properties se:mustUnderstand="1">
      <expiresAt>20070620T170000</expiresAt>
      <sentAt>20070619T170000</sentAt>
    </properties>
    <Stream se:mustUnderstand="1">
      <streamId>uid:2744e4e1-2b48-43e8-b441-42745f280d53\111</streamId>
      <current>2</current>
    </Stream>
    <Msmq xmlns="msmq.namespace.xml">
      <Class>0</Class>
      <Priority>0</Priority>
      <Correlation>AAAAAAAAAAAAAAAAAAAAAAAAAAA=</Correlation>
      <App>0</App>
      <BodyType>0</BodyType>
      <HashAlgorithm>32772</HashAlgorithm>
      <SourceQmGuid>6a74a825-57b2-43e5-9d34-f1d8b2b8950a</SourceQmGuid>
      <TTrq>20070620T170000</TTrq>
    </Msmq>
  </se:Header>
  <se:Body></se:Body>
</se:Envelope>--MSMQ - SOAP boundary, 1672
Content-Type: application/octet-stream
Content-Length: 9
Content-Id: body@ff3af301-3196-497a-a918-72147c871a13

Message 0--MSMQ - SOAP boundary, 1672--
```

Stream receipt message (ordering ack) acknowledging the second message:

```
POST /msmq/private$/receipts?senderstream=xrntv HTTP/1.1
Host: machine1
Content-Type: text/xml
Content-Length: 993
SOAPAction: "MSMQMessage"
Proxy-Accept: NonInteractiveClient
```

```
<se:Envelope xmlns:se="http://schemas.xmlsoap.org/soap/envelope/"
xmlns="http://schemas.xmlsoap.org/srmp/">
  <se:Header>
    <path xmlns="http://schemas.xmlsoap.org/rp/" se:mustUnderstand="1">
      <action>MSMQ:QM Ordering Ack</action>
      <to>http://machine1/MSMQ/private$/receipts?SenderStream=XRntV</to>
      <rev>
        <via>http://machine2/msmq/private$/tsimpleq</via>
      </rev>
      <id>uuid:26643@32221eda-9376-46df-b6ed-783091123831</id>
    </path>
    <properties se:mustUnderstand="1">
      <expiresAt>20380119T031407</expiresAt>
      <sentAt>20070619T165959</sentAt>
    </properties>
    <streamReceipt>
      <streamId>uid:2744e4e1-2b48-43e8-b441-42745f280d53\111</streamId>
      <lastOrdinal>2</lastOrdinal>
    </streamReceipt>
    <Msmq xmlns="msmq.namespace.xml">
      <Class>255</Class>
      <Priority>0</Priority>
      <Correlation>AAAAAAAAAAAAAAAAAAAAAAAAAAA=</Correlation>
      <App>0</App>
      <BodyType>0</BodyType>
      <HashAlgorithm>0</HashAlgorithm>
      <SourceQmGuid>32221eda-9376-46df-b6ed-783091123831</SourceQmGuid>
      <TTrq>20070623T165959</TTrq>
    </Msmq>
  </se:Header>
  <se:Body></se:Body>
</se:Envelope>
```

Third message in stream.

```
POST /msmq/private$/tsimpleq HTTP/1.1
Host: machine2
Content-Type: multipart/related; boundary="MSMQ - SOAP boundary, 1672";
type=text/xml
Content-Length: 1524
SOAPAction: "MSMQMessage"
Proxy-Accept: NonInteractiveClient

--MSMQ - SOAP boundary, 1672
Content-Type: text/xml; charset=UTF-8
Content-Length: 1240

<se:Envelope xmlns:se="http://schemas.xmlsoap.org/soap/envelope/"
xmlns="http://schemas.xmlsoap.org/srmp/">
  <se:Header>
    <path xmlns="http://schemas.xmlsoap.org/rp/" se:mustUnderstand="1">
      <action>MSMQ:mqsender label</action>
      <to>http://machine2/msmq/private$/tsimpleq</to>
      <id>uuid:1@00000000-0000-0000-0000-000000000008</id>
    </path>
    <properties se:mustUnderstand="1">
      <expiresAt>20070620T170001</expiresAt>
      <sentAt>20070619T170001</sentAt>
    </properties>
    <Stream se:mustUnderstand="1">
      <streamId>uid:2744e4e1-2b48-43e8-b441-42745f280d53\111</streamId>
```

```
      <current>3</current>
      <end/>
    </Stream>
    <Msmq xmlns="msmq.namespace.xml">
      <Class>0</Class>
      <Priority>0</Priority>
      <Correlation>AAAAAAAAAAAAAAAAAAAAAAAAAAA=</Correlation>
      <App>0</App>
      <BodyType>0</BodyType>
      <HashAlgorithm>32772</HashAlgorithm>
      <SourceQmGuid>f306d4c7-be95-419a-9e3b-8c751d04b1e9</SourceQmGuid>
      <TTrq>20070620T170001</TTrq>
    </Msmq>
  </se:Header>
  <se:Body></se:Body>
</se:Envelope>--MSMQ - SOAP boundary, 1672
Content-Type: application/octet-stream
Content-Length: 12
Content-Id: body@ff3af301-3196-497a-a918-72147c871a13

Last Message--MSMQ - SOAP boundary, 1672--
```

Stream receipt message (ordering ack) acknowledging the third and last message:

```
POST /msmq/private$/receipts?senderstream=xrntv HTTP/1.1
Host: machine1
Content-Type: text/xml
Content-Length: 993
SOAPAction: "MSMQMessage"
Proxy-Accept: NonInteractiveClient

<se:Envelope xmlns:se="http://schemas.xmlsoap.org/soap/envelope/"
xmlns="http://schemas.xmlsoap.org/srmp/">
  <se:Header>
    <path xmlns="http://schemas.xmlsoap.org/rp/" se:mustUnderstand="1">
    <action>MSMQ:QM Ordering Ack</action>
      <to>http://machine1/MSMQ/private$/receipts?SenderStream=XRntV</to>
      <rev>
        <via>http://machine2/msmq/private$/tsimpleq</via>
      </rev>
      <id>uuid:26645@32221eda-9376-46df-b6ed-783091123831</id>
    </path>
    <properties se:mustUnderstand="1">
      <expiresAt>20380119T031407</expiresAt>
      <sentAt>20070619T170001</sentAt>
    </properties>
    <streamReceipt>
      <streamId>uid:2744e4e1-2b48-43e8-b441-42745f280d53\111</streamId>
      <lastOrdinal>3</lastOrdinal>
    </streamReceipt>
    <Msmq xmlns="msmq.namespace.xml">
      <Class>255</Class>
      <Priority>0</Priority>
      <Correlation>AAAAAAAAAAAAAAAAAAAAAAAAAAA=</Correlation>
      <App>0</App>
      <BodyType>0</BodyType>
      <HashAlgorithm>0</HashAlgorithm>
      <SourceQmGuid>32221eda-9376-46df-b6ed-783091123831</SourceQmGuid>
      <TTrq>20070623T170001</TTrq>
    </Msmq>
  </se:Header>
  <se:Body></se:Body>
```

```
</se:Envelope>
```

4.5 PGM Example

The following is an example of a simple SRMP message using the PGM transport instead of TCP.

```
POST 234.1.1.1:8001 HTTP/1.1
Host: 234.1.1.1:8001
Content-Type: multipart/related; boundary="MSMQ - SOAP boundary, 292";
type=text/xml
Content-Length: 1067
SOAPAction: "MSMQMessage"
Proxy-Accept: NonInteractiveClient

--MSMQ - SOAP boundary, 292
Content-Type: text/xml; charset=UTF-8
Content-Length: 779
<se:Envelope xmlns:se="http://schemas.xmlsoap.org/soap/envelope/"
xmlns="http://schemas.xmlsoap.org/srmp/">
  <se:Header>
    <path xmlns="http://schemas.xmlsoap.org/rp/" se:mustUnderstand="1">
      <action>MSMQ:mqsender label</action>
      <to>MSMQ:MULTICAST=234.1.1.1:8001</to>
      <id>uuid:4494@32221eda-9376-46df-b6ed-783091123831</id>
    </path>
    <properties se:mustUnderstand="1">
      <expiresAt>20380119T031407</expiresAt>
      <sentAt>20070608T023719</sentAt>
    </properties>
    <Msmq xmlns="msmq.namespace.xml">
      <Class>0</Class>
      <Priority>3</Priority>
      <Correlation>AAAAAAAAAAAAAAAAAAAAAAAAAAA=</Correlation>
      <App>0</App>
      <BodyType>8</BodyType>
      <HashAlgorithm>32772</HashAlgorithm>
      <SourceQmGuid>32221eda-9376-46df-b6ed-783091123831</SourceQmGuid>
      <TTrq>20070612T023719</TTrq>
    </Msmq>
  </se:Header>
  <se:Body></se:Body>
</se:Envelope>--MSMQ - SOAP boundary, 292
Content-Type: application/octet-stream
Content-Length: 20
Content-Id: body@32221eda-9376-46df-b6ed-783091123831

a.r.d.e.n.w.h.i.t.e.--MSMQ - SOAP boundary, 292--
```

5 Security

5.1 Security Considerations for Implementers

The SRMP relies on HTTPS for security. HTTPS relies on SSL. The message syntax of SRMP does not contain elements for authentication and encryption.

6 Appendix A: Windows Behavior

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

- Windows XP
- Windows Server 2003
- Windows Vista

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 Windows does not follow the prescription.

<1> Section 2.1: PGM-based SRMP messages are received directly into the MSMQ service, completely bypassing the Web server. Normal SRMP messages using TCP are passed to MSMQ unprocessed. With both TCP and PGM, the HTTP 1.1 message is passed into MSMQ, where the entire message is processed.

<2> Section 2.1.1: In addition to following the SHOULDs in the required sections of [RFC3208], Windows also follows the MAYs. Exceptions that affect interoperability are noted as follows:

- Section 2.1 (Transmit windows advance – 3rd MAY): Windows does not delay the window advancement based on NAK-silence.
- Section 5.1.1 (MAY): Windows does not use any definition of TXW_MAX_RTE other than that defined in the RFC.
- Section 5.2 (SHOULD): Windows does not do any NAK storm detection. If a NAK request arrives for a sequence in the current window, an NCF will be sent followed by RDATA.
- Section 5.3 (SHOULD): Windows does not track propagation delays in computing the RDATA servicing delay time. This delay time prior to servicing the RDATA request is computed according to the formula in section 2.1.1.1 of this specification.
- Section 6.1 (MAY in last paragraph): Windows will only deliver data to the receiver application in the order sent, not in the order received.
- Section 6.3 (Transmitting a NAK – MAY): The multicasting of the NAK happens irrespective of whether the PGM parent is directly connected or not.
- Section 9.4 (OPT_JOIN): Windows does not send the Late Joining Option.
- Section 9.6 (both MAYs): Windows does not use the SYN notification either to notify the application about the start of the stream, or to notify a late joining receiver whether it missed any data or not from the start of the stream.
- Section 9.6.1 (MAY): Windows provides statistical information to applications about the data stream such as rate and loss information, but does not provide any abstractions of the stream based on SYNs.
- Section 9.7.1 (MAY in last paragraph): Windows does not provide any direct notification of the receipt of a FIN, but the session will terminate gracefully once all the data has been delivered successfully to the application.

- Section 9.8.1 (MAY): Once the Windows receiver receives an OPT_RST, it terminates the session immediately without attempting to recover any pending data.
- Section 9.8.2 (SHOULDs): A Windows source sends an OPT_RST only if the sends are cancelled; otherwise the session is closed gracefully. When the sends are cancelled, the session will be terminated immediately by the source, and subsequent NAKs will not be processed.
- Section 16.1 (3rd MAY): Windows does not reset TXW_ADV_IVL_TMR if NAKs are received.
- Section 16.4 (MAY): Windows does not provide any other method of advancing the transmit window other than as specified in section 2.1.1.1 of this specification.

<3> Section 2.1.1.1: Windows allows a system administrator to configure alternate values of TXW_SECS and TXW_MAX_RTE.

<4> Section 2.2.4.1: If the value included in the <action> element is not prefixed by "MSMQ:" then Windows ignores the contents of the element entirely.

<5> Section 2.2.4.1: If the value included in the <action> element is not prefixed by "MSMQ:" then Windows treats the contents of the <action> element as though it contained only the string "MSMQ:" when sending delivery and commitment receipts.

<6> Section 2.2.4.3: Windows XP follows this rule, but processes the message if the GUIDs do not match.

<7> Section 2.2.4.4: The original WS-Routing specification [WS-ROUTING] states, "Multiple <via> elements MAY be present, but all except the first <via> element MUST be ignored." SRMP imposes a stricter standard.

<8> Section 2.2.4.4: In the case of an empty <via> element, Windows XP allows the form "<via/>" but does not allow the form "<via></via>".

<9> Section 3.1.1.2: Each Windows XP client and server generates a unique GUID upon setup and stores it durably as a binary value under the HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\MSMQ\Parameters\MachineCache\QMId registry key.

<10> Section 3.1.1.2: The Windows XP implementation converts the SRMP message internally into a **UserMessage** packet. For more information, see [MS-MQQB] section 2.2.8.

<11> Section 3.1.1.2: The Windows XP implementation persistently stores up to the last 10,000 **MessageIdentifer** values. Any values greater than 30 minutes old are discarded.

<12> Section 3.1.2: The Windows XP implementation creates a new sequence when there are no unacknowledged stream messages.

<13> Section 3.1.3.1: Windows XP discards regular, durable, and stream messages after all outstanding receipts have been received.

<14> Section 3.1.3.2: The Windows XP implementation waits a minimum of 500 milliseconds before sending a stream receipt message. If a new stream message is received during this period, then the stream receipt is delayed another 500 milliseconds. This process continues and can delay the stream receipt up to 10 seconds.

<15> Section 3.1.3.2: Windows XP discards regular, durable, and stream messages after all outstanding receipts have been received.

<16> Section 3.1.3.3: Windows XP discards regular, durable, and stream messages after all outstanding receipts have been received.

<17> Section 3.2.2: The Windows XP default time-out is 30 seconds. The time-out grows as the number of sequential time-outs occurs. The first, second, and third time-outs have periods of 30 seconds. The fourth, fifth, and sixth time-outs are 5 minutes. The seventh, eighth, and ninth time-outs are 30 minutes, and thereafter, the time-out period is 6 hours.

<18> Section 3.3.1: The default setting is 20,000 milliseconds on a local area network (LAN). The Microsoft implementation provides a registry key at HKEY_LOCAL_MACHINE\software\microsoft\msmq\parameters\RoundTripDelay that the client may use to specify additional time beyond the default value.

<19> Section 3.3.1: Each Windows XP client and server generates a unique GUID upon setup and stores it durably as a binary value under the HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\MSMQ\Parameters\MachineCache\QMId registry key.

<20> Section 3.5.2.3: Windows XP copies the message to the system queue journal. The queue journal is a system-generated queue and is implementation-dependent.

<21> Section 3.5.3.2: The Windows XP implementation creates a new sequence when there are no unacknowledged transactional messages.

<22> Section 3.5.3.3: Windows XP stores a copy of the message in the local dead-letter queue on failure to deliver. Stream messages are copied to the local transactional dead-letter queue. The dead-letter queue is a system-generated queue and is implementation-dependent.

<23> Section 3.5.4: The Windows XP dead-letter queue, transactional dead-letter queue, and queue journal are named "Deadletter$", "XactDeadletter", and "Journal$", respectively.

<24> Section 3.5.5.2: Windows XP parses the value in the <id> element and separates it into its components, the message index and the source QM GUID.

<25> Section 3.5.5.3: Windows XP stores a copy of the message in the local dead-letter queue on failure to deliver. Stream messages are copied to the local transactional dead-letter queue. The dead-letter queue is a system-generated queue and is implementation-dependent.

<26> Section 3.5.5.4: The Windows XP implementation waits a minimum of 500 milliseconds before sending a stream receiptmessage. If a new stream message is received during this period, then the stream receipt is delayed another 500 milliseconds. This process continues and can delay the stream receipt up to 10 seconds.

7 Index