Web Services Reliable Messaging (WS-ReliableMessaging) Version 1.2

Committee Draft 02

20 November 2008

Specification URIs:
This Version:
http://docs.oasis-open.org/ws-rx/wsrm/200702/wsrm-1.2-spec-cd-02.pdf
http://docs.oasis-open.org/ws-rx/wsrm/200702/wsrm-1.2-spec-cd-02.html
http://docs.oasis-open.org/ws-rx/wsrm/200702/wsrm-1.2-spec-cd-02.doc (Authoritative)

Previous Version:
http://docs.oasis-open.org/ws-rx/wsrm/200702/wsrm-1.2-spec-cs-01.pdf
http://docs.oasis-open.org/ws-rx/wsrm/200702/wsrm-1.2-spec-cs-01.html
http://docs.oasis-open.org/ws-rx/wsrm/200702/wsrm-1.2-spec-cs-01.doc

Latest Version:
http://docs.oasis-open.org/ws-rx/wsrm/v1.2/wsrm.pdf
http://docs.oasis-open.org/ws-rx/wsrm/v1.2/wsrm.html
http://docs.oasis-open.org/ws-rx/wsrm/v1.2/wsrm.doc

Technical Committee:
OASIS Web Services Reliable Exchange (WS-RX) TC

Chairs:
Paul Fremantle <paul@wso2.com>
Sanjay Patil <sanjay.patil@sap.com>

Editors:
Doug Davis, IBM <dug@us.ibm.com>
Anish Karmarkar, Oracle <Anish.Karmarkar@oracle.com>
Gilbert Pilz, BEA <gpilz@bea.com>
Steve Winkler, SAP <steve.winkler@sap.com>
Ümit Yalçinalp, SAP <umit.yalcinalp@sap.com>

Related Work:
This specification replaces or supercedes:

- WS-ReliableMessaging v1.1

Declared XML Namespaces:
http://docs.oasis-open.org/ws-rx/wsrm/200702

Abstract:
This specification (WS-ReliableMessaging) describes a protocol that allows messages to be transferred reliably between nodes implementing this protocol in the presence of software component, system, or network failures. The protocol is described in this specification in a transport-independent manner allowing it to be implemented using different network technologies. To support interoperable Web services, a SOAP binding is defined within this specification.

The protocol defined in this specification depends upon other Web services specifications for the identification of service endpoint addresses and policies. How these are identified and retrieved are detailed within those specifications and are out of scope for this document.

By using the XML [XML], SOAP [SOAP 1.1], [SOAP 1.2] and WSDL [WSDL 1.1] extensibility model, SOAP-based and WSDL-based specifications are designed to be composed with each other to define a rich Web services environment. As such, WS-ReliableMessaging by itself does not define all the features required for a complete messaging solution. WS-ReliableMessaging is a building block that is used in conjunction with other specifications and application-specific protocols to accommodate a wide variety of requirements and scenarios related to the operation of distributed Web services.

Status:

This document was last revised or approved by the WS-RX Technical Committee on the above date. The level of approval is also listed above. Check the "Latest Version" or "Latest Approved Version" location noted above for possible later revisions of this document.

Technical Committee members should send comments on this specification to the Technical Committee's email list. Others should send comments to the Technical Committee by using the "Send A Comment" button on the Technical Committee's web page at http://www.oasis-open.org/committees/ws-rx/.

For information on whether any patents have been disclosed that may be essential to implementing this specification, and any offers of patent licensing terms, please refer to the Intellectual Property Rights section of the Technical Committee web page (http://www.oasis-open.org/committees/ws-rx/ipr.php).

The non-normative errata page for this specification is located at http://www.oasis-open.org/committees/ws-rx/.

Notices

Table of Contents

1 Introduction

It is often a requirement for two Web services that wish to communicate to do so reliably in the presence of software component, system, or network failures. The primary goal of this specification is to create a modular mechanism for reliable transfer of messages. It defines a messaging protocol to identify, track, and manage the reliable transfer of messages between a source and a destination. It also defines a SOAP binding that is required for interoperability. Additional bindings can be defined.

This mechanism is extensible allowing additional functionality, such as security, to be tightly integrated. This specification integrates with and complements the WS-Security [WS-Security], WS-Policy [WS-Policy], and other Web services specifications. Combined, these allow for a broad range of reliable, secure messaging options.

1.1 Terminology

The keywords "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [KEYWORDS].

This specification uses the following syntax to define normative outlines for messages:

- The syntax appears as an XML instance, but values in italics indicate data types instead of values.
- Characters are appended to elements and attributes to indicate cardinality:
    - "?" (0 or 1)
    - "*" (0 or more)
    - "+" (1 or more)
- The character "|" is used to indicate a choice between alternatives.
- The characters "[" and "]" are used to indicate that contained items are to be treated as a group with respect to cardinality or choice.
- An ellipsis (i.e. "...") indicates a point of extensibility that allows other child or attribute content specified in this document. Additional children elements and/or attributes MAY be added at the indicated extension points but they MUST NOT contradict the semantics of the parent and/or owner, respectively. If an extension is not recognized it SHOULD be ignored.
- XML namespace prefixes (see section 1.4) are used to indicate the namespace of the element being defined.

Elements and Attributes defined by this specification are referred to in the text of this document using XPath 1.0 [XPath_10] expressions. Extensibility points are referred to using an extended version of this syntax:

- An element extensibility point is referred to using {any} in place of the element name. This indicates that any element name can be used, from any namespace other than the wsrm: namespace.
- An attribute extensibility point is referred to using @{any} in place of the attribute name. This indicates that any attribute name can be used, from any namespace other than the wsrm: namespace.

1.2 Normative References

[KEYWORDS]	S. Bradner, "Key words for use in RFCs to Indicate Requirement Levels," RFC 2119, Harvard University, March 1997 http://www.ietf.org/rfc/rfc2119.txt
[WS-RM Policy]	OASIS WS-RX Technical Committee Draft, "Web Services Reliable Messaging Policy Assertion(WS-RM Policy)," November 2008 http://docs.oasis-open.org/ws-rx/wsrmp/v1.2/wsrmp.pdf
[SOAP 1.1]	W3C Note, "SOAP: Simple Object Access Protocol 1.1," 08 May 2000. http://www.w3.org/TR/2000/NOTE-SOAP-20000508/
[SOAP 1.2]	W3C Recommendation, "SOAP Version 1.2 Part 1: Messaging Framework" June 2003. http://www.w3.org/TR/2003/REC-soap12-part1-20030624/
[URI]	T. Berners-Lee, R. Fielding, L. Masinter, "Uniform Resource Identifiers (URI): Generic Syntax," RFC 3986, MIT/LCS, U.C. Irvine, Xerox Corporation, January 2005. http://ietf.org/rfc/rfc3986
[UUID]	P. Leach, M. Mealling, R. Salz, "A Universally Unique IDentifier (UUID) URN Namespace," RFC 4122, Microsoft, Refactored Networks - LLC, DataPower Technology Inc, July 2005 http://www.ietf.org/rfc/rfc4122.txt
[XML]	W3C Recommendation, "Extensible Markup Language (XML) 1.0 (Fourth Edition)", September 2006. http://www.w3.org/TR/REC-xml/
[XML-ns]	W3C Recommendation, "Namespaces in XML," 14 January 1999. http://www.w3.org/TR/1999/REC-xml-names-19990114/
[XML-Schema Part1]	W3C Recommendation, "XML Schema Part 1: Structures," October 2004. http://www.w3.org/TR/xmlschema-1/
[XML-Schema Part2]	W3C Recommendation, "XML Schema Part 2: Datatypes," October 2004. http://www.w3.org/TR/xmlschema-2/
[XPATH 1.0]	W3C Recommendation, "XML Path Language (XPath) Version 1.0," 16 November 1999. http://www.w3.org/TR/xpath
[WSDL 1.1]	W3C Note, "Web Services Description Language (WSDL 1.1)," 15 March 2001. http://www.w3.org/TR/2001/NOTE-wsdl-20010315
[WS-Addressing]	W3C Recommendation, "Web Services Addressing 1.0 – Core," May 2006. http://www.w3.org/TR/2006/REC-ws-addr-core-20060509/ W3C Recommendation, "Web Services Addressing 1.0 – SOAP Binding," May 2006 http://www.w3.org/TR/2006/REC-ws-addr-soap-20060509/

1.3 Non-Normative References

[BSP 1.0]	WS-I Working Group Draft. "Basic Security Profile Version 1.0," August 2006 http://www.ws-i.org/Profiles/BasicSecurityProfile-1.0.html
[RDDL 2.0]	Jonathan Borden, Tim Bray, eds. "Resource Directory Description Language (RDDL) 2.0," January 2004 http://www.openhealth.org/RDDL/20040118/rddl-20040118.html
[RFC 2617]	J. Franks, P. Hallam-Baker, J. Hostetler, S. Lawrence, P. Leach, A. Loutonen, L. Stewart, "HTTP Authentication: Basic and Digest Access Authentication," June

1999.
http://www.ietf.org/rfc/rfc2617.txt

[RFC 4346] T. Dierks, E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.1," April 2006.
http://www.ietf.org/rfc/rfc4346.txt

[WS-Policy] W3C Recommendation, "Web Services Policy 1.5 - Framework," September 2007.
http://www.w3.org/TR/2007/REC-ws-policy-20070904

[WS-PolicyAttachment] W3C Recommendation, "Web Services Policy 1.5 - Attachment," September 2007.
http://www.w3.org/TR/2007/REC-ws-policy-attach-20070904

[WS-Security] Anthony Nadalin, Chris Kaler, Phillip Hallam-Baker, Ronald Monzillo, eds. "OASIS Web Services Security: SOAP Message Security 1.0 (WS-Security 2004)", OASIS Standard 200401, March 2004.
http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0.pdf

Anthony Nadalin, Chris Kaler, Phillip Hallam-Baker, Ronald Monzillo, eds. "OASIS Web Services Se-curity: SOAP Message Security 1.1 (WS-Security 2004)", OASIS Standard 200602, February 2006.
http://docs.oasis-open.org/wss/v1.1/wss-v1.1-spec-os-SOAPMessageSecurity.pdf

[RTTM] V. Jacobson, R. Braden, D. Borman, "TCP Extensions for High Performance", RFC 1323, May 1992.
http://www.rfc-editor.org/rfc/rfc1323.txt

[SecurityPolicy] OASIS WS-SX Technical Committee Editor Draft, "WS-SecurityPolicy 1.3"
http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200802

[SecureConversation] OASIS WS-SX Technical Committee Editor Draft, "WS-SecureConversation 1.4"
http://docs.oasis-open.org/ws-sx/ws-secureconversation/200512

[Trust] OASIS WS-SX Technical Committee Editor Draft, "WS-Trust 1.4"
http://docs.oasis-open.org/ws-sx/ws-trust/200802

1.4 Namespace

The XML namespace [XML-ns] URI that MUST be used by implementations of this specification is:

```
http://docs.oasis-open.org/ws-rx/wsrm/200702
```

Dereferencing the above URI will produce the Resource Directory Description Language [RDDL 2.0] document that describes this namespace.

Table 1 lists the XML namespaces that are used in this specification. The choice of any namespace prefix is arbitrary and not semantically significant.

Table 1

Prefix	Namespace
S	(Either SOAP 1.1 or 1.2)
S11	http://schemas.xmlsoap.org/soap/envelope/
S12	http://www.w3.org/2003/05/soap-envelope
wsrm	http://docs.oasis-open.org/ws-rx/wsrm/200702
wsa	http://www.w3.org/2005/08/addressing
wsam	http://www.w3.org/2007/05/addressing/metadata

wsse	http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd
xs	http://www.w3.org/2001/XMLSchema

The normative schema for WS-ReliableMessaging can be found linked from the namespace document that is located at the namespace URI specified above.

All sections explicitly noted as examples are informational and are not to be considered normative.

1.5 Conformance

An implementation is not conformant with this specification if it fails to satisfy one or more of the MUST or REQUIRED level requirements defined herein. A SOAP Node MUST NOT use the XML namespace identifier for this specification (listed in section 1.4) within SOAP Envelopes unless it is conformant with this specification.

Normative text within this specification takes precedence over normative outlines, which in turn take precedence over the XML Schema [XML Schema Part 1, Part 2] descriptions.

2 Reliable Messaging Model

Many errors can interrupt a conversation. Messages can be lost, duplicated or reordered. Further the host systems can experience failures and lose volatile state.

The WS-ReliableMessaging specification defines an interoperable protocol that enables a Reliable Messaging (RM) Source to accurately determine the disposition of each message it Transmits as perceived by the RM Destination, so as to allow it to resolve any in-doubt status regarding receipt of the message Transmitted. The protocol also enables an RM Destination to efficiently determine which of those messages it Receives have been previously Received, enabling it to filter out duplicate message transmissions caused by the retransmission, by the RM Source, of an unacknowledged message. It also enables an RM Destination to Deliver the messages it Receives to the Application Destination in the order in which they were sent by an Application Source, in the event that they are Received out of order. Note that this specification places no restriction on the scope of the RM Source or RM Destination entities. For example, either can span multiple WSDL Ports or Endpoints.

The protocol enables the implementation of a broad range of reliability features which include ordered Delivery, duplicate elimination, and guaranteed receipt. The protocol can also be implemented with a range of robustness characteristics ranging from in-memory persistence that is scoped to a single process lifetime, to replicated durable storage that is recoverable in all but the most extreme circumstances. It is expected that the Endpoints will implement as many or as few of these reliability characteristics as necessary for the correct operation of the application using the protocol. Regardless of which of the reliability features is enabled, the wire protocol does not change.

Figure 1 below illustrates the entities and events in a simple reliable exchange of messages. First, the Application Source Sends a message for reliable transfer. The Reliable Messaging Source accepts the message and Transmits it one or more times. After accepting the message, the RM Destination Acknowledges it. Finally, the RM Destination Delivers the message to the Application Destination. The exact roles the entities play and the complete meaning of the events will be defined throughout this specification.

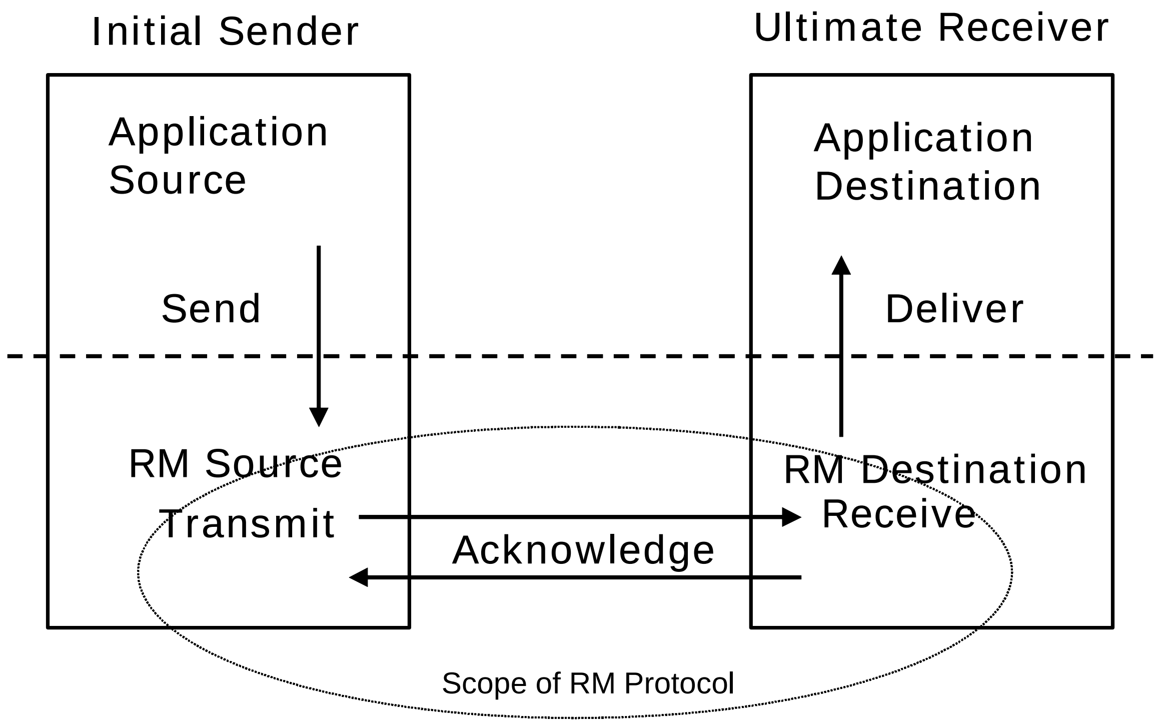

Figure 1: Reliable Messaging Model

2.1 Glossary

The following definitions are used throughout this specification:

Accept: The act of qualifying a message by the RM Destination such that it becomes eligible for Delivery and acknowledgement**.**

Acknowledgement: The communication from the RM Destination to the RM Source indicating the successful receipt of a message.

Acknowledgement Message: A message containing a `SequenceAcknowledgement` header block. Acknowledgement Messages may or may not contain a SOAP body.

Acknowledgement Request: A message containing an `AckRequested` header. Acknowledgement Requests may or may not contain a SOAP body.

Application Destination: The Endpoint to which a message is Delivered.

Application Source: The Endpoint that Sends a message.

Back-channel: When the underlying transport provides a mechanism to return a transport-protocol specific response, capable of carrying a SOAP message, without initiating a new connection, this specification refers to this mechanism as a back-channel.

Deliver: The act of transferring responsibility for a message from the RM Destination to the Application Destination.

Endpoint: As defined in the WS-Addressing specification [WS-Addressing]; a Web service Endpoint is a (referenceable) entity, processor, or resource to which Web service messages can be addressed. Endpoint references (EPRs) convey the information needed to address a Web service Endpoint.

Receive: The act of reading a message from a network connection and accepting it.

RM Destination: The Endpoint that Receives messages Transmitted reliably from an RM Source.

RM Protocol Header Block: One of `Sequence`, `SequenceAcknowledgement`, or `AckRequested`.

RM Source: The Endpoint that Transmits messages reliably to an RM Destination.

Send: The act of transferring a message from the Application Source to the RM Source for reliable transfer.

Sequence Lifecycle Message: A message that contains one of: `CreateSequence`, `CreateSequenceResponse`, `CloseSequence`, `CloseSequenceResponse`, `TerminateSequence`, `TerminateSequenceResponse` as the child element of the SOAP body element.

Sequence Traffic Message: A message containing a `Sequence` header block.

Transmit: The act of writing a message to a network connection.

2.2 Protocol Preconditions

The correct operation of the protocol requires that a number of preconditions MUST be established prior to the processing of the initial sequenced message:

- For any single message exchange the RM Source MUST have an endpoint reference that uniquely identifies the RM Destination Endpoint.
- The RM Source MUST have successfully created a Sequence with the RM Destination.
- The RM Source MUST be capable of formulating messages that adhere to the RM Destination's policies.
- If a secure exchange of messages is REQUIRED, then the RM Source and RM Destination MUST have a security context.

2.3 Protocol Invariants

During the lifetime of a Sequence, the following invariants are REQUIRED for correctness:

- The RM Source MUST assign each message within a Sequence a message number (defined below) beginning at 1 and increasing by exactly 1 for each subsequent message. These numbers MUST be assigned in the same order in which messages are sent by the Application Source.
- Within every Acknowledgement Message it issues, the RM Destination MUST include one or more `AcknowledgementRange` child elements that contain, in their collective ranges, the message number of every message accepted by the RM Destination. The RM Destination MUST exclude, in the `AcknowledgementRange` elements, the message numbers of any messages it has not accepted. If no messages have been received the RM Destination MUST return `None` instead of an `AcknowledgementRange(s)`. The RM Destination MAY transmit a `Nack` for a specific message or messages instead of an `AcknowledgementRange(s)`.
- While the Sequence is not closed or terminated, the RM Source SHOULD retransmit unacknowledged messages.

2.4 Delivery Assurances

This section defines a number of Delivery Assurance assertions, which can be supported by RM Sources and RM Destinations. These assertions can be specified as policy assertions using the WS-Policy framework [WS-Policy]. For details on this see the WSRM Policy specification [WS-RM Policy].

`AtLeastOnce`

Each message is to be delivered at least once, or else an error MUST be raised by the RM Source and/or RM Destination. The requirement on an RM Source is that it SHOULD retry transmission of every message sent by the Application Source until it receives an acknowledgement from the RM Destination. The requirement on the RM Destination is that it SHOULD retry the transfer to the Application Destination of any message that it accepts from the RM Source, until that message has been successfully delivered. There is no requirement for the RM Destination to apply duplicate message filtering.

AtMostOnce

Each message is to be delivered at most once. The RM Source MAY retry transmission of unacknowledged messages, but is NOT REQUIRED to do so. The requirement on the RM Destination is that it MUST filter out duplicate messages, i.e. that it MUST NOT deliver a duplicate of a message that has already been delivered.

ExactlyOnce

Each message is to be delivered exactly once; if a message cannot be delivered then an error MUST be raised by the RM Source and/or RM Destination. The requirement on an RM Source is that it SHOULD retry transmission of every message sent by the Application Source until it receives an acknowledgement from the RM Destination. The requirement on the RM Destination is that it SHOULD retry the transfer to the Application Destination of any message that it accepts from the RM Source until that message has been successfully delivered, and that it MUST NOT deliver a duplicate of a message that has already been delivered.

InOrder

Messages from each individual Sequence are to be delivered in the same order they have been sent by the Application Source. The requirement on an RM Source is that it MUST ensure that the ordinal position of each message in the Sequence (as indicated by a message Sequence number) is consistent with the order in which the messages have been sent from the Application Source. The requirement on the RM Destination is that it MUST deliver received messages for each Sequence in the order indicated by the message numbering. This DeliveryAssurance can be used in combination with any of the AtLeastOnce, AtMostOnce or ExactlyOnce assertions, and the requirements of those assertions MUST also be met. In particular if the AtLeastOnce or ExactlyOnce assertion applies and the RM Destination detects a gap in the Sequence then the RM Destination MUST NOT deliver any subsequent messages from that Sequence until the missing messages are received or until the Sequence is closed.

2.5 Example Message Exchange

Figure 2 illustrates a possible message exchange between two reliable messaging Endpoints A and B.

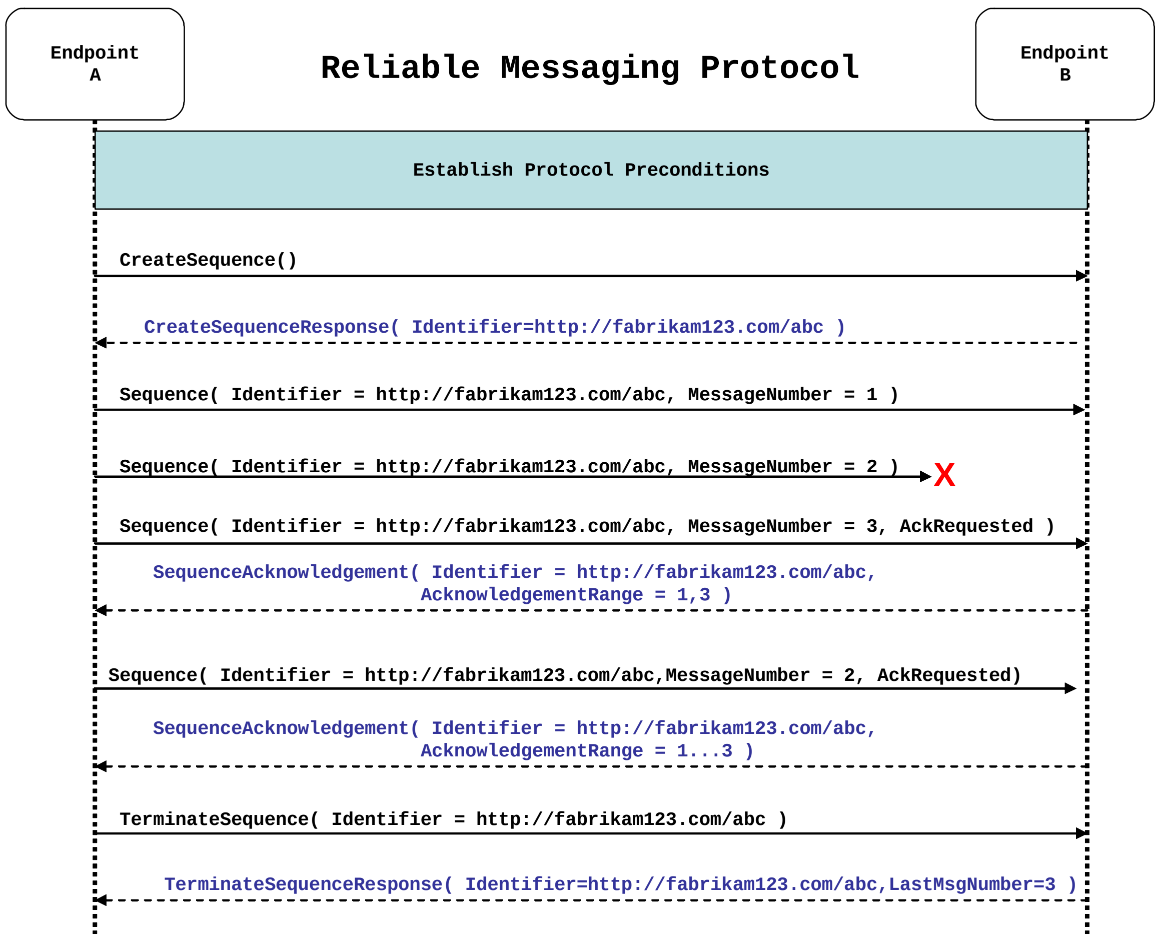
Endpoint A
Reliable Messaging Protocol
Endpoint B
Establish Protocol Preconditions
CreateSequence()
CreateSequenceResponse(Identifier=http://fabrikam123.com/abc)
Sequence(Identifier = http://fabrikam123.com/abc, MessageNumber = 1)
Sequence(Identifier = http://fabrikam123.com/abc, MessageNumber = 2)
X
Sequence(Identifier = http://fabrikam123.com/abc, MessageNumber = 3, AckRequested)
SequenceAcknowledgement(Identifier = http://fabrikam123.com/abc, AcknowledgementRange = 1,3)
Sequence(Identifier = http://fabrikam123.com/abc,MessageNumber = 2, AckRequested)
SequenceAcknowledgement(Identifier = http://fabrikam123.com/abc, AcknowledgementRange = 1...3)
TerminateSequence(Identifier = http://fabrikam123.com/abc)
TerminateSequenceResponse(Identifier=http://fabrikam123.com/abc,LastMsgNumber=3)

Figure 2: The WS-ReliableMessaging Protocol

1. The protocol preconditions are established. These include policy exchange, endpoint resolution, and establishing trust.
2. The RM Source requests creation of a new Sequence.
3. The RM Destination creates a new Sequence and returns its unique `Identifier`.
4. The RM Source begins Transmitting messages in the Sequence beginning with `MessageNumber` 1. In the figure above, the RM Source sends 3 messages in the Sequence.
5. The 2nd message in the Sequence is lost in transit.
6. The 3rd message is the last in this Sequence and the RM Source includes an `AckRequested` header to ensure that it gets a timely `SequenceAcknowledgement` for the Sequence.
7. The RM Destination acknowledges receipt of message numbers 1 and 3 as a result of receiving the RM Source's `AckRequested` header.
8. The RM Source retransmits the unacknowledged message with `MessageNumber` 2. This is a new message from the perspective of the underlying transport, but it has the same Sequence `Identifier` and `MessageNumber` so the RM Destination can recognize it as a duplicate of the earlier message, in case the original and retransmitted messages are both Received. The RM Source includes an `AckRequested` header in the retransmitted message so the RM Destination will expedite an acknowledgement.

9. The RM Destination Receives the second transmission of the message with `MessageNumber` 2 and acknowledges receipt of message numbers 1, 2, and 3.

10. The RM Source Receives this Acknowledgement and sends a `TerminateSequence` message to the RM Destination indicating that the Sequence is completed. The `TerminateSequence` message indicates that message number 3 was the last message in the Sequence. The RM Destination then reclaims any resources associated with the Sequence.

11. The RM Destination Receives the `TerminateSequence` message indicating that the RM Source will not be sending any more messages. The RM Destination sends a `TerminateSequenceResponse` message to the RM Source and reclaims any resources associated with the Sequence.

The RM Source will expect to Receive Acknowledgements from the RM Destination during the course of a message exchange at occasions described in section 3 below. Should an Acknowledgement not be Received in a timely fashion, the RM Source MUST re-transmit the message since either the message or the associated Acknowledgement might have been lost. Since the nature and dynamic characteristics of the underlying transport and potential intermediaries are unknown in the general case, the timing of re-transmissions cannot be specified. Additionally, over-aggressive re-transmissions have been demonstrated to cause transport or intermediary flooding which are counterproductive to the intention of providing a reliable exchange of messages. Consequently, implementers are encouraged to utilize adaptive mechanisms that dynamically adjust re-transmission time and the back-off intervals that are appropriate to the nature of the transports and intermediaries envisioned. For the case of TCP/IP transports, a mechanism similar to that described as RTTM in RFC 1323 [RTTM] SHOULD be considered.

Now that the basic model has been outlined, the details of the elements used in this protocol are now provided in section 3.

3 RM Protocol Elements

The following sub-sections define the various RM protocol elements, and prescribe their usage by a conformant implementations.

3.1 Considerations on the Use of Extensibility Points

The following protocol elements define extensibility points at various places. Implementations MAY add child elements and/or attributes at the indicated extension points but MUST NOT contradict the semantics of the parent and/or owner, respectively. If a receiver does not recognize an extension, the receiver SHOULD ignore the extension.

3.2 Considerations on the Use of "Piggy-Backing"

Some RM Protocol Header Blocks may be added to messages that are targeted to the same Endpoint to which those headers are to be sent (a concept often referred to as "piggy-backing"), thus saving the overhead of an additional message exchange. Reference parameters MUST be considered when determining whether two EPRs are targeted to the same Endpoint. The determination of if and when a Header Block will be piggy-backed onto another message is made by the entity (RM Source or RM Destination) that is sending the header. In order to ensure optimal and successful processing of RM Sequences, endpoints that receive RM-related messages SHOULD be prepared to process RM Protocol Header Blocks that are included in any message it receives. See the sections that define each RM Protocol Header Block to know which ones may be considered for piggy-backing.

3.3 Composition with WS-Addressing

When the RM protocol, defined in this specification, is composed with the WS-Addressing specification, the following rules prescribe the constraints on the value of the `wsa:Action` header:

1. When an Endpoint generates a message that carries an RM protocol element, that is defined in the following sections, in the body of a SOAP envelope that Endpoint MUST include in that envelope a `wsa:Action` SOAP header block whose value is an IRI that is a concatenation of the WS-RM namespace URI, followed by a "/", followed by the value of the local name of the child element of the SOAP body . For example, for a Sequence creation request message as described in section 3.4 below, the value of the `wsa:Action` IRI would be:

```
http://docs.oasis-open.org/ws-rx/wsrm/200702/CreateSequence
```

2. When an Endpoint generates an Acknowledgement Message that has no element content in the SOAP body, then the value of the `wsa:Action` IRI MUST be:

```
http://docs.oasis-open.org/ws-rx/wsrm/200702/SequenceAcknowledgement
```

3. When an Endpoint generates an Acknowledgement Request that has no element content in the SOAP body, then the value of the `wsa:Action` IRI MUST be:

```
http://docs.oasis-open.org/ws-rx/wsrm/200702/AckRequested
```

4. When an Endpoint generates an RM fault as defined in section 4 below, the value of the `wsa:Action` IRI MUST be as defined in section 4 below.

3.4 Sequence Creation

The RM Source MUST request creation of an outbound Sequence by sending a `CreateSequence` element in the body of a message to the RM Destination which in turn responds either with a message containing `CreateSequenceResponse` or a `CreateSequenceRefused` fault. The RM Source MAY include an offer to create an inbound Sequence within the `CreateSequence` message. This offer is either accepted or rejected by the RM Destination in the `CreateSequenceResponse` message.

The SOAP version used for the `CreateSequence` message SHOULD be used for all subsequent messages in or for that Sequence, sent by either the RM Source or the RM Destination.

The following exemplar defines the `CreateSequence` syntax:

```
<wsrm:CreateSequence ...>
    <wsrm:AcksTo> wsa:EndpointReferenceType </wsrm:AcksTo>
    <wsrm:Expires ...> xs:duration </wsrm:Expires> ?
    <wsrm:Offer ...>
        <wsrm:Identifier ...> xs:anyURI </wsrm:Identifier>
        <wsrm:Endpoint> wsa:EndpointReferenceType </wsrm:Endpoint>
        <wsrm:Expires ...> xs:duration </wsrm:Expires> ?
        <wsrm:IncompleteSequenceBehavior>
            wsrm:IncompleteSequenceBehaviorType
        </wsrm:IncompleteSequenceBehavior> ?
        ...
    </wsrm:Offer> ?
    ...
</wsrm:CreateSequence>
```

The following describes the content model of the `CreateSequence` element.

`/wsrm:CreateSequence`

This element requests creation of a new Sequence between the RM Source that sends it, and the RM Destination to which it is sent. The RM Source MUST NOT send this element as a header block. The RM Destination MUST respond either with a `CreateSequenceResponse` response message or a `CreateSequenceRefused` fault.

`/wsrm:CreateSequence/wsrm:AcksTo`

The RM Source MUST include this element in any `CreateSequence` message it sends. This element is of type `wsa:EndpointReferenceType` (as specified by WS-Addressing). It specifies the endpoint reference to which messages containing `SequenceAcknowledgement` header blocks and faults related to the created Sequence are to be sent, unless otherwise noted in this specification (for example, see section 3.5).

Implementations MUST NOT use an endpoint reference in the `AcksTo` element that would prevent the sending of Sequence Acknowledgements back to the RM Source. For example, using the WS-Addressing "http://www.w3.org/2005/08/addressing/none" IRI would make it impossible for the RM Destination to ever send Sequence Acknowledgements.

`/wsrm:CreateSequence/wsrm:Expires`

This element, if present, of type `xs:duration` specifies the RM Source's requested duration for the Sequence. The RM Destination MAY either accept the requested duration or assign a lesser value of its choosing. A value of "PT0S" indicates that the Sequence will never expire. Absence of the element indicates an implied value of "PT0S".

`/wsrm:CreateSequence/wsrm:Expires/@{any}`

This is an extensibility mechanism to allow additional attributes, based on schemas, to be added to the element.

`/wsrm:CreateSequence/wsrm:Offer`

This element, if present, enables an RM Source to offer a corresponding Sequence for the reliable exchange of messages Transmitted from RM Destination to RM Source.

`/wsrm:CreateSequence/wsrm:Offer/wsrm:Identifier`

The RM Source MUST set the value of this element to an absolute URI (conformant with RFC3986 [URI]) that uniquely identifies the offered Sequence.

`/wsrm:CreateSequence/wsrm:Offer/wsrm:Identifier/@{any}`

This is an extensibility mechanism to allow additional attributes, based on schemas, to be added to the element.

`/wsrm:CreateSequence/wsrm:Offer/wsrm:Endpoint`

An RM Source MUST include this element, of type `wsa:EndpointReferenceType` (as specified by WS-Addressing). This element specifies the endpoint reference to which Sequence Lifecycle Messages, Acknowledgement Requests, and fault messages related to the offered Sequence are to be sent.

Implementations MUST NOT use an endpoint reference in the Endpoint element that would prevent the sending of Sequence Lifecycle Message, etc. For example, using the WS-Addressing "http://www.w3.org/2005/08/addressing/none" IRI would make it impossible for the RM Destination to ever send Sequence Lifecycle Messages (e.g. `TerminateSequence`) to the RM Source for the offered Sequence.

The offer of an Endpoint containing the "http://www.w3.org/2005/08/addressing/anonymous" IRI as its address is problematic due to the inability of a source to connect to this address and retry unacknowledged messages (as described in section 2.3). Note that this specification does not define any mechanisms for providing this assurance. In the absence of an extension that addresses this issue, an RM Destination MUST NOT accept (via the `/wsrm:CreateSequenceResponse/wsrm:Accept` element described below) an offer that contains the "http://www.w3.org/2005/08/addressing/anonymous" IRI as its `address`.

`/wsrm:CreateSequence/wsrm:Offer/wsrm:Expires`

This element, if present, of type `xs:duration` specifies the duration for the offered Sequence. A value of "PT0S" indicates that the offered Sequence will never expire. Absence of the element indicates an implied value of "PT0S".

`/wsrm:CreateSequence/wsrm:Offer/wsrm:Expires/@{any}`

This is an extensibility mechanism to allow additional attributes, based on schemas, to be added to the element.

`/wsrm:CreateSequence/wsrm:Offer/wsrm:IncompleteSequenceBehavior`

This element, if present, specifies the behavior that the destination will exhibit upon the closure or termination of an incomplete Sequence. For the purposes of defining the values used, the term "discard" refers to behavior equivalent to the Application Destination never processing a particular message.

A value of “`DiscardEntireSequence`” indicates that the entire Sequence MUST be discarded if the Sequence is closed, or terminated, when there are one or more gaps in the final `SequenceAcknowledgement`.

A value of “`DiscardFollowingFirstGap`” indicates that messages in the Sequence beyond the first gap MUST be discarded when there are one or more gaps in the final `SequenceAcknowledgement`.

The default value of "NoDiscard" indicates that no acknowledged messages in the Sequence will be discarded.

/wsrm:CreateSequence/wsrm:Offer/{any}

This is an extensibility mechanism to allow different (extensible) types of information, based on a schema, to be passed.

/wsrm:CreateSequence/wsrm:Offer/@{any}

This is an extensibility mechanism to allow additional attributes, based on schemas, to be added to the element.

/wsrm:CreateSequence/{any}

This is an extensibility mechanism to allow different (extensible) types of information, based on a schema, to be passed.

/wsrm:CreateSequence/@{any}

This is an extensibility mechanism to allow additional attributes, based on schemas, to be added to the element.

A `CreateSequenceResponse` is sent in the body of a response message by an RM Destination in response to receipt of a `CreateSequence` request message. It carries the `Identifier` of the created Sequence and indicates that the RM Source can begin sending messages in the context of the identified Sequence.

The following exemplar defines the `CreateSequenceResponse` syntax:

```
<wsrm:CreateSequenceResponse ...>
    <wsrm:Identifier ...> xs:anyURI </wsrm:Identifier>
    <wsrm:Expires ...> xs:duration </wsrm:Expires> ?
    <wsrm:IncompleteSequenceBehavior>
        wsrm:IncompleteSequenceBehaviorType
    </wsrm:IncompleteSequenceBehavior> ?
    <wsrm:Accept ...>
        <wsrm:AcksTo> wsa:EndpointReferenceType </wsrm:AcksTo>
        ...
    </wsrm:Accept> ?
    ...
</wsrm:CreateSequenceResponse>
```

The following describes the content model of the `CreateSequenceResponse` element.

/wsrm:CreateSequenceResponse

This element is sent in the body of the response message in response to a `CreateSequence request message`. It indicates that the RM Destination has created a new Sequence at the request of the RM Source. The RM Destination MUST NOT send this element as a header block.

/wsrm:CreateSequenceResponse/wsrm:Identifier

The RM Destination MUST include this element within any `CreateSequenceResponse` message it sends. The RM Destination MUST set the value of this element to the absolute URI (conformant with RFC3986) that uniquely identifies the Sequence that has been created by the RM Destination.

/wsrm:CreateSequenceResponse/wsrm:Identifier/@{any}

This is an extensibility mechanism to allow additional attributes, based on schemas, to be added to the element.

/wsrm:CreateSequenceResponse/wsrm:Expires

This element, if present, of type `xs:duration` accepts or refines the RM Source's requested duration for the Sequence. It specifies the amount of time after which any resources associated with the Sequence SHOULD be reclaimed thus causing the Sequence to be silently terminated. At the RM Destination this duration is measured from a point proximate to Sequence creation and at the RM Source this duration is measured from a point approximate to the successful processing of the `CreateSequenceResponse`. A value of "PT0S" indicates that the Sequence will never expire. Absence of the element indicates an implied value of "PT0S". The RM Destination MUST set the value of this element to be equal to or less than the value requested by the RM Source in the corresponding `CreateSequence` message.

`/wsrm:CreateSequenceResponse/wsrm:Expires/@{any}`

This is an extensibility mechanism to allow additional attributes, based on schemas, to be added to the element.

`/wsrm:CreateSequenceResponse/wsrm:IncompleteSequenceBehavior`

This element, if present, specifies the behavior that the destination will exhibit upon the closure or termination of an incomplete Sequence. For the purposes of defining the values used, the term "discard" refers to behavior equivalent to the Application Destination never processing a particular message.

A value of “`DiscardEntireSequence`” indicates that the entire Sequence MUST be discarded if the Sequence is closed, or terminated, when there are one or more gaps in the final `SequenceAcknowledgement`.

A value of “`DiscardFollowingFirstGap`” indicates that messages in the Sequence beyond the first gap MUST be discarded when there are one or more gaps in the final `SequenceAcknowledgement`.

The default value of “`NoDiscard`” indicates that no acknowledged messages in the Sequence will be discarded.

`/wsrm:CreateSequenceResponse/wsrm:Accept`

This element, if present, enables an RM Destination to accept the offer of a corresponding Sequence for the reliable exchange of messages Transmitted from RM Destination to RM Source.

Note: If a `CreateSequenceResponse` is returned without a child `Accept` in response to a `CreateSequence` that did contain a child `Offer`, then the RM Source MAY immediately reclaim any resources associated with the unused offered Sequence.

`/wsrm:CreateSequenceResponse/wsrm:Accept/wsrm:AcksTo`

The RM Destination MUST include this element, of type `wsa:EndpointReferenceType` (as specified by WS-Addressing). It specifies the endpoint reference to which messages containing `SequenceAcknowledgement` header blocks and faults related to the created Sequence are to be sent, unless otherwise noted in this specification (for example, see section3.5).

Implementations MUST NOT use an endpoint reference in the `AcksTo` element that would prevent the sending of Sequence Acknowledgements back to the RM Source. For example, using the WS-Addressing "http://www.w3.org/2005/08/addressing/none" IRI would make it impossible for the RM Destination to ever send Sequence Acknowledgements.

`/wsrm:CreateSequenceResponse/wsrm:Accept/{any}`

This is an extensibility mechanism to allow different (extensible) types of information, based on a schema, to be passed.

`/wsrm:CreateSequenceResponse/wsrm:Accept/@{any}`

This is an extensibility mechanism to allow additional attributes, based on schemas, to be added to the element.

`/wsrm:CreateSequenceResponse/{any}`

This is an extensibility mechanism to allow different (extensible) types of information, based on a schema, to be passed.

`/wsrm:CreateSequenceResponse/@{any}`

This is an extensibility mechanism to allow additional attributes, based on schemas, to be added to the element.

3.5 Closing A Sequence

There are times during the use of an RM Sequence that the RM Source or RM Destination will wish to discontinue using a Sequence. Simply terminating the Sequence discards the state managed by the RM Destination, leaving the RM Source unaware of the final ranges of messages that were successfully transferred to the RM Destination. To ensure that the Sequence ends with a known final state either the RM Source or RM Destination MAY choose to close the Sequence before terminating it.

If the RM Source wishes to close the Sequence, then it sends a `CloseSequence` element, in the body of a message, to the RM Destination. This message indicates that the RM Destination MUST NOT accept any new messages for the specified Sequence, other than those already accepted at the time the `CloseSequence` element is interpreted by the RM Destination. Upon receipt of this message, or subsequent to the RM Destination closing the Sequence of its own volition, the RM Destination MUST include a final `SequenceAcknowledgement` (within which the RM Destination MUST include the `Final` element) header block on any messages associated with the Sequence destined to the RM Source, including the `CloseSequenceResponse` message or on any Sequence fault Transmitted to the RM Source.

To allow the RM Destination to determine if it has received all of the messages in a Sequence, the RM Source SHOULD include the `LastMsgNumber` element in any `CloseSequence` messages it sends. The RM Destination can use this information, for example, to implement the behavior indicated by `/wsrm:CreateSequenceResponse/wsrm:IncompleteSequenceBehavior`. The value of the `LastMsgNumber` element MUST be the same in all the `CloseSequence` messages for the closing Sequence.

If the RM Destination decides to close a Sequence of its own volition, it MAY inform the RM Source of this event by sending a `CloseSequence` element, in the body of a message, to the `AcksTo` EPR of that Sequence. The RM Destination MUST include a final `SequenceAcknowledgement` (within which the RM Destination MUST include the `Final` element) header block in this message and any subsequent messages associated with the Sequence destined to the RM Source.

While the RM Destination MUST NOT accept any new messages for the specified Sequence it MUST still process Sequence Lifecyle Messages and Acknowledgement Requests. For example, it MUST respond to `AckRequested`, `TerminateSequence` as well as `CloseSequence` messages. Note, subsequent `CloseSequence` messages have no effect on the state of the Sequence.

In the case where the RM Destination wishes to discontinue use of a Sequence it is RECOMMENDED that it close the Sequence. Please see `Final` and the `SequenceClosed` fault. Whenever possible the `SequenceClosed` fault SHOULD be used in place of the `SequenceTerminated` fault to allow the RM Source to still Receive Acknowledgements.

The following exemplar defines the `CloseSequence` syntax:

```
<wsrm:CloseSequence ...>
    <wsrm:Identifier ...> xs:anyURI </wsrm:Identifier>
    <wsrm:LastMsgNumber> wsrm:MessageNumberType </wsrm:LastMsgNumber> ?
```

```
   ...
</wsrm:CloseSequence>
```

The following describes the content model of the `CloseSequence` element.

/wsrm:CloseSequence

This element MAY be sent by an RM Source to indicate that the RM Destination MUST NOT accept any new messages for this Sequence This element MAY also be sent by an RM Destination to indicate that it will not accept any new messages for this Sequence.

/wsrm:CloseSequence/wsrm:Identifier

The RM Source or RM Destination MUST include this element in any `CloseSequence` messages it sends. The RM Source or RM Destination MUST set the value of this element to the absolute URI (conformant with RFC3986) of the closing Sequence.

/wsrm:CloseSequence/wsrm:LastMsgNumber

The RM Source SHOULD include this element in any `CloseSequence` message it sends. The `LastMsgNumber` element specifies the highest assigned message number of all the Sequence Traffic Messages for the closing Sequence.

/wsrm:CloseSequence/wsrm:Identifier/@{any}

This is an extensibility mechanism to allow additional attributes, based on schemas, to be added to the element.

/wsrm:CloseSequence/{any}

This is an extensibility mechanism to allow different (extensible) types of information, based on a schema, to be passed.

/wsrm:CloseSequence/@{any}

This is an extensibility mechanism to allow additional attributes, based on schemas, to be added to the element.

A `CloseSequenceResponse` is sent in the body of a message in response to receipt of a `CloseSequence` request message. It indicates that the responder has closed the Sequence.

The following exemplar defines the `CloseSequenceResponse` syntax:

```
<wsrm:CloseSequenceResponse ...>
    <wsrm:Identifier ...> xs:anyURI </wsrm:Identifier>
    ...
</wsrm:CloseSequenceResponse>
```

The following describes the content model of the `CloseSequenceResponse` element.

/wsrm:CloseSequenceResponse

This element is sent in the body of a message in response to receipt of a `CloseSequence` request message. It indicates that the responder has closed the Sequence.

/wsrm:CloseSequenceResponse/wsrm:Identifier

The responder (RM Source or RM Destination) MUST include this element in any `CloseSequenceResponse` message it sends. The responder MUST set the value of this element to the absolute URI (conformant with RFC3986) of the closing Sequence.

/wsrm:CloseSequenceResponse/wsrm:Identifier/@{any}

This is an extensibility mechanism to allow additional attributes, based on schemas, to be added to the element.

`/wsrm:CloseSequenceResponse/{any}`

> This is an extensibility mechanism to allow different (extensible) types of information, based on a schema, to be passed.

`/wsrm:CloseSequenceResponse/@{any}`

> This is an extensibility mechanism to allow additional attributes, based on schemas, to be added to the element.

3.6 Sequence Termination

When the RM Source has completed its use of the Sequence it sends a `TerminateSequence` element, in the body of a message, to the RM Destination to indicate that the Sequence is complete and that it will not be sending any further messages related to the Sequence. The RM Destination can safely reclaim any resources associated with the Sequence upon receipt of the `TerminateSequence` message. Under normal usage the RM Source will complete its use of the Sequence when all of the messages in the Sequence have been acknowledged. However, the RM Source is free to Terminate or Close a Sequence at any time regardless of the acknowledgement state of the messages.

To allow the RM Destination to determine if it has received all of the messages in a Sequence, the RM Source SHOULD include the `LastMsgNumber` element in any `TerminateSequence` messages it sends. The RM Destination can use this information, for example, to implement the behavior indicated by `/wsrm:CreateSequenceResponse/wsrm:IncompleteSequenceBehavior`. The value of the `LastMsgNumber` element in the `TerminateSequence` message MUST be equal to the value of the `LastMsgNumber` element in any `CloseSequence` message(s) sent by the RM Source for the same Sequence.

If the RM Destination decides to terminate a Sequence of its own volition, it MAY inform the RM Source of this event by sending a `TerminateSequence` element, in the body of a message, to the `AcksTo` EPR for that Sequence. The RM Destination MUST include a final `SequenceAcknowledgement` (within which the RM Destination MUST include the `Final` element) header block in this message.

The following exemplar defines the `TerminateSequence` syntax:

```
<wsrm:TerminateSequence ...>
    <wsrm:Identifier ...> xs:anyURI </wsrm:Identifier>
    <wsrm:LastMsgNumber> wsrm:MessageNumberType </wsrm:LastMsgNumber> ?
    ...
</wsrm:TerminateSequence>
```

The following describes the content model of the `TerminateSequence` element.

`/wsrm:TerminateSequence`

> This element MAY be sent by an RM Source to indicate it has completed its use of the Sequence. It indicates that the RM Destination can safely reclaim any resources related to the identified Sequence. The RM Source MUST NOT send this element as a header block. The RM Source MAY retransmit this element. Once this element is sent, other than this element, the RM Source MUST NOT send any additional message to the RM Destination referencing this Sequence.

> This element MAY also be sent by the RM Destination to indicate that it has unilaterally terminated the Sequence. Upon sending this message the RM Destination MUST NOT accept any additional messages (with the exception of the corresponding `TerminateSequenceResponse`) for this Sequence. Upon receipt of a `TerminateSequence` the RM Source MUST NOT send any additional messages (with the exception of the corresponding `TerminateSequenceResponse`) for this Sequence.

`/wsrm:TerminateSequence/wsrm:Identifier`

The RM Source or RM Destination MUST include this element in any `TerminateSequence` message it sends. The RM Source or RM Destination MUST set the value of this element to the absolute URI (conformant with RFC3986) of the terminating Sequence.

/wsrm:TerminateSequence/wsrm:LastMsgNumber

The RM Source SHOULD include this element in any `TerminateSequence` message it sends. The `LastMsgNumber` element specifies the highest assigned message number of all the Sequence Traffic Messages for the terminating Sequence.

/wsrm:TerminateSequence/wsrm:Identifier/@{any}

This is an extensibility mechanism to allow additional attributes, based on schemas, to be added to the element.

/wsrm:TerminateSequence/{any}

This is an extensibility mechanism to allow different (extensible) types of information, based on a schema, to be passed.

/wsrm:TerminateSequence/@{any}

This is an extensibility mechanism to allow additional attributes, based on schemas, to be added to the element.

A `TerminateSequenceResponse` is sent in the body of a message in response to receipt of a `TerminateSequence` request message. It indicates that responder has terminated the Sequence.

The following exemplar defines the `TerminateSequenceResponse` syntax:

```
<wsrm:TerminateSequenceResponse ...>
    <wsrm:Identifier ...> xs:anyURI </wsrm:Identifier>
    ...
</wsrm:TerminateSequenceResponse>
```

The following describes the content model of the `TerminateSequence` element.

/wsrm:TerminateSequenceResponse

This element is sent in the body of a message in response to receipt of a `TerminateSequence` request message. It indicates that the responder has terminated the Sequence. The responder MUST NOT send this element as a header block.

/wsrm:TerminateSequenceResponse/wsrm:Identifier

The responder (RM Source or RM Destination) MUST include this element in any `TerminateSequenceResponse` message it sends. The responder MUST set the value of this element to the absolute URI (conformant with RFC3986) of the terminating Sequence.

/wsrm:TerminateSequenceResponse/wsrm:Identifier/@{any}

This is an extensibility mechanism to allow additional attributes, based on schemas, to be added to the element.

/wsrm:TerminateSequenceResponse/{any}

This is an extensibility mechanism to allow different (extensible) types of information, based on a schema, to be passed.

/wsrm:TerminateSequenceResponse/@{any}

This is an extensibility mechanism to allow additional attributes, based on schemas, to be added to the element.

On receipt of a `TerminateSequence` message the receiver (RM Source or RM Destination) MUST respond with a corresponding `TerminateSequenceResponse` message or generate a fault `UnknownSequenceFault` if the Sequence is not known.

3.7 Sequences

The RM protocol uses a Sequence header block to track and manage the reliable transfer of messages. The RM Source MUST include a `Sequence` header block in all messages for which reliable transfer is REQUIRED. The RM Source MUST identify Sequences with unique `Identifier` elements and the RM Source MUST assign each message within a Sequence a `MessageNumber` element that increments by 1 from an initial value of 1. These values are contained within a `Sequence` header block accompanying each message being transferred in the context of a Sequence.

The RM Source MUST NOT include more than one `Sequence` header block in any message.

A following exemplar defines its syntax:

```
<wsrm:Sequence ...>
    <wsrm:Identifier ...> xs:anyURI </wsrm:Identifier>
    <wsrm:MessageNumber> wsrm:MessageNumberType </wsrm:MessageNumber>
    ...
</wsrm:Sequence>
```

The following describes the content model of the `Sequence` header block.

`/wsrm:Sequence`

This protocol element associates the message in which it is contained with a previously established RM Sequence. It contains the Sequence's unique `Identifier` and the containing message's ordinal position within that Sequence. The RM Destination MUST understand the `Sequence` header block. The RM Source MUST assign a `mustUnderstand` attribute with a value 1/true (from the namespace corresponding to the version of SOAP to which the `Sequence` SOAP header block is bound) to the `Sequence` header block element.

`/wsrm:Sequence/wsrm:Identifier`

An RM Source that includes a `Sequence` header block in a SOAP envelope MUST include this element in that header block. The RM Source MUST set the value of this element to the absolute URI (conformant with RFC3986) that uniquely identifies the Sequence.

`/wsrm:Sequence/wsrm:Identifier/@{any}`

This is an extensibility mechanism to allow additional attributes, based on schemas, to be added to the element.

`/wsrm:Sequence/wsrm:MessageNumber`

The RM Source MUST include this element within any `Sequence` headers it creates. This element is of type `MessageNumberType.` It represents the ordinal position of the message within a Sequence. Sequence message numbers start at 1 and monotonically increase by 1 throughout the Sequence. See section 4.5 for Message Number Rollover fault.

`/wsrm:Sequence/{any}`

This is an extensibility mechanism to allow different (extensible) types of information, based on a schema, to be passed.

`/wsrm:Sequence/@{any}`

This is an extensibility mechanism to allow additional attributes, based on schemas, to be added to the element.

The following example illustrates a `Sequence` header block.

```
<wsrm:Sequence>
    <wsrm:Identifier>http://example.com/abc</wsrm:Identifier>
    <wsrm:MessageNumber>10</wsrm:MessageNumber>
</wsrm:Sequence>
```

3.8 Request Acknowledgement

The purpose of the `AckRequested` header block is to signal to the RM Destination that the RM Source is requesting that a `SequenceAcknowledgement` be sent.

The RM Source MAY request an Acknowledgement Message from the RM Destination at any time by independently transmitting an `AckRequested` header block (i.e. as a header of a SOAP envelope with an empty body). Alternatively the RM Source MAY include an `AckRequested` header block in any message targeted to the RM Destination. The RM Destination SHOULD process `AckRequested` header blocks that are included in any message it receives. If a non-mustUnderstand fault occurs when processing an `AckRequested` header block that was piggy-backed, a fault MUST be generated, but the processing of the original message MUST NOT be affected.

An RM Destination that Receives a message that contains an `AckRequested` header block MUST send a message containing a `SequenceAcknowledgement` header block to the `AcksTo` endpoint reference (see section 3.4) for a known Sequence or else generate an `UnknownSequence` fault. It is RECOMMENDED that the RM Destination return a `AcknowledgementRange` or `None` element instead of a `Nack` element (see section 3.9).

The following exemplar defines its syntax:

```
<wsrm:AckRequested ...>
    <wsrm:Identifier ...> xs:anyURI </wsrm:Identifier>
    ...
</wsrm:AckRequested>
```

The following describes the content model of the `AckRequested` header block.

`/wsrm:AckRequested`

This element requests an Acknowledgement for the identified Sequence.

`/wsrm:AckRequested/wsrm:Identifier`

An RM Source that includes an `AckRequested` header block in a SOAP envelope MUST include this element in that header block. The RM Source MUST set the value of this element to the absolute URI, (conformant with RFC3986), that uniquely identifies the Sequence to which the request applies.

`/wsrm:AckRequested/wsrm:Identifier/@{any}`

This is an extensibility mechanism to allow additional attributes, based on schemas, to be added to the element.

`/wsrm:AckRequested/{any}`

This is an extensibility mechanism to allow different (extensible) types of information, based on a schema, to be passed.

`/wsrm:AckRequested/@{any}`

This is an extensibility mechanism to allow additional attributes, based on schemas, to be added to the element.

3.9 Sequence Acknowledgement

The RM Destination informs the RM Source of successful message receipt using a `SequenceAcknowledgement` header block. Acknowledgements can be explicitly requested using the `AckRequested` directive (see section 3.8).

The RM Destination MAY Transmit the `SequenceAcknowledgement` header block independently (i.e. as a header of a SOAP envelope with an empty body). Alternatively, an RM Destination MAY include a `SequenceAcknowledgement` header block on any SOAP envelope targeted to the endpoint referenced by the `AcksTo` EPR. The RM Source SHOULD process `SequenceAcknowledgement` header blocks that are included in any message it receives. If a non-mustUnderstand fault occurs when processing a `SequenceAcknowledgement` header that was piggy-backed, a fault MUST be generated, but the processing of the original message MUST NOT be affected.

During creation of a Sequence the RM Source MAY specify the WS-Addressing anonymous IRI as the `address` of the `AcksTo` EPR for that Sequence. When the RM Source specifies the WS-Addressing anonymous IRI as the `address` of the `AcksTo` EPR, the RM Destination MUST Transmit any `SequenceAcknowledgement` headers for the created Sequence in a SOAP envelope to be Transmitted on the protocol binding-specific back-channel. Such a channel is provided by the context of a Received message containing a SOAP envelope that contains a `Sequence` header block and/or an `AckRequested` header block for that same Sequence `Identifier`. When the RM Destination receives an `AckRequested` header, and the `AcksTo` EPR for that Sequence is the WS-Addressing anonymous IRI, the RM Destination SHOULD respond on the protocol binding-specific back-channel provided by the Received message containing the `AckRequested` header block.

The following exemplar defines its syntax:

```
<wsrm:SequenceAcknowledgement ...>
    <wsrm:Identifier ...> xs:anyURI </wsrm:Identifier>
    [ [ [ <wsrm:AcknowledgementRange ...
            Upper="wsrm:MessageNumberType"
            Lower="wsrm:MessageNumberType"/> +
        | <wsrm:None/> ]
        <wsrm:Final/> ? ]
    | <wsrm:Nack> wsrm:MessageNumberType </wsrm:Nack> + ]

    ...
</wsrm:SequenceAcknowledgement>
```

The following describes the content model of the `SequenceAcknowledgement` header block.

/wsrm:SequenceAcknowledgement

This element contains the Sequence Acknowledgement information.

/wsrm:SequenceAcknowledgement/wsrm:Identifier

An RM Destination that includes a `SequenceAcknowledgement` header block in a SOAP envelope MUST include this element in that header block. The RM Destination MUST set the value of this element to the absolute URI (conformant with RFC3986) that uniquely identifies the Sequence. The RM Destination MUST NOT include multiple `SequenceAcknowledgement` header blocks that share the same value for `Identifier` within the same SOAP envelope.

/wsrm:SequenceAcknowledgement/wsrm:Identifier/@{any}

This is an extensibility mechanism to allow additional attributes, based on schemas, to be added to the element.

/wsrm:SequenceAcknowledgement/wsrm:AcknowledgementRange

The RM Destination MAY include one or more instances of this element within a `SequenceAcknowledgement` header block. It contains a range of Sequence message numbers successfully accepted by the RM Destination. The ranges MUST NOT overlap. The RM Destination MUST NOT include this element if a sibling `Nack` or `None` element is also present as a child of `SequenceAcknowledgement`.

`/wsrm:SequenceAcknowledgement/wsrm:AcknowledgementRange/@Upper`

The RM Destination MUST set the value of this attribute equal to the message number of the highest contiguous message in a Sequence range accepted by the RM Destination.

`/wsrm:SequenceAcknowledgement/wsrm:AcknowledgementRange/@Lower`

The RM Destination MUST set the value of this attribute equal to the message number of the lowest contiguous message in a Sequence range accepted by the RM Destination.

`/wsrm:SequenceAcknowledgement/wsrm:AcknowledgementRange/@{any}`

This is an extensibility mechanism to allow additional attributes, based on schemas, to be added to the element.

`/wsrm:SequenceAcknowledgement/wsrm:None`

The RM Destination MUST include this element within a `SequenceAcknowledgement` header block if the RM Destination has not accepted any messages for the specified Sequence. The RM Destination MUST NOT include this element if a sibling `AcknowledgementRange` or `Nack` element is also present as a child of the `SequenceAcknowledgement`.

`/wsrm:SequenceAcknowledgement/wsrm:Final`

The RM Destination MAY include this element within a `SequenceAcknowledgement` header block. This element indicates that the RM Destination is not receiving new messages for the specified Sequence. The RM Source can be assured that the ranges of messages acknowledged by this `SequenceAcknowledgement` header block will not change in the future. The RM Destination MUST include this element when the Sequence is closed. The RM Destination MUST NOT include this element when sending a `Nack`; it can only be used when sending `AcknowledgementRange` elements or a `None`.

`/wsrm:SequenceAcknowledgement/wsrm:Nack`

The RM Destination MAY include this element within a `SequenceAcknowledgement` header block. If used, the RM Destination MUST set the value of this element to a `MessageNumberType` representing the `MessageNumber` of an unreceived message in a Sequence. The RM Destination MUST NOT include a `Nack` element if a sibling `AcknowledgementRange` or `None` element is also present as a child of `SequenceAcknowledgement`. Upon the receipt of a `Nack`, an RM Source SHOULD retransmit the message identified by the `Nack`. The RM Destination MUST NOT issue a `SequenceAcknowledgement` containing a `Nack` for a message that it has previously acknowledged within an `AcknowledgementRange`. The RM Source SHOULD ignore a `SequenceAcknowledgement` containing a `Nack` for a message that has previously been acknowledged within an `AcknowledgementRange`.

`/wsrm:SequenceAcknowledgement/{any}`

This is an extensibility mechanism to allow different (extensible) types of information, based on a schema, to be passed.

`/wsrm:SequenceAcknowledgement/@{any}`

This is an extensibility mechanism to allow additional attributes, based on schemas, to be added to the element.

The following examples illustrate `SequenceAcknowledgement` elements:

- Message numbers 1...10 inclusive in a Sequence have been accepted by the RM Destination.

```
<wsrm:SequenceAcknowledgement>
    <wsrm:Identifier>http://example.com/abc</wsrm:Identifier>
    <wsrm:AcknowledgementRange Upper="10" Lower="1"/>
</wsrm:SequenceAcknowledgement>
```

- Message numbers 1..2, 4..6, and 8..10 inclusive in a Sequence have been accepted by the RM Destination, messages 3 and 7 have not been accepted.

```
<wsrm:SequenceAcknowledgement>
    <wsrm:Identifier>http://example.com/abc</wsrm:Identifier>
    <wsrm:AcknowledgementRange Upper="2" Lower="1"/>
    <wsrm:AcknowledgementRange Upper="6" Lower="4"/>
    <wsrm:AcknowledgementRange Upper="10" Lower="8"/>
</wsrm:SequenceAcknowledgement>
```

- Message number 3 in a Sequence has not been accepted by the RM Destination.

```
<wsrm:SequenceAcknowledgement>
    <wsrm:Identifier>http://example.com/abc</wsrm:Identifier>
    <wsrm:Nack>3</wsrm:Nack>
</wsrm:SequenceAcknowledgement>
```

4 Faults

Faults for the `CreateSequence` message exchange are treated as defined in WS-Addressing. Create Sequence Refused is a possible fault reply for this operation. Unknown Sequence is a fault generated by Endpoints when messages carrying RM header blocks targeted at unrecognized or terminated Sequences are detected. `WSRMRequired` is a fault generated by an RM Destination that requires the use of WS-RM on a Received message that did not use the protocol. All other faults in this section relate to known Sequences. Destinations that generate faults related to known Sequences SHOULD transmit those faults. If transmitted, such faults MUST be transmitted to the same [destination] as Acknowledgement messages.

Entities that generate WS-ReliableMessaging faults MUST include as the [action] property the default fault action IRI defined below. The value from the W3C Recommendation is below for informational purposes:

```
http://docs.oasis-open.org/ws-rx/wsrm/200702/fault
```

The faults defined in this section are generated if the condition stated in the preamble is met. Fault handling rules are defined in section 6 of WS-Addressing SOAP Binding.

The definitions of faults use the following properties:

[Code] The fault code.

[Subcode] The fault subcode.

[Reason] The English language reason element.

[Detail] The detail element(s). If absent, no detail element is defined for the fault. If more than one detail element is defined for a fault, implementations MUST include the elements in the order that they are specified.

Entities that generate WS-ReliableMessaging faults MUST set the [Code] property to either "Sender" or "Receiver". These properties are serialized into text XML as follows:

SOAP Version	Sender	Receiver
SOAP 1.1	S11:Client	S11:Server
SOAP 1.2	S:Sender	S:Receiver

The properties above bind to a SOAP 1.2 fault as follows:

```
<S:Envelope>
 <S:Header>
   <wsa:Action>
      http://docs.oasis-open.org/ws-rx/wsrm/200702/fault
   </wsa:Action>
   <!-- Headers elided for brevity.  -->
 </S:Header>
 <S:Body>
  <S:Fault>
   <S:Code>
     <S:Value> [Code] </S:Value>
     <S:Subcode>
      <S:Value> [Subcode] </S:Value>
     </S:Subcode>
   </S:Code>
   <S:Reason>
     <S:Text xml:lang="en"> [Reason] </S:Text>
   </S:Reason>
   <S:Detail>
     [Detail]
```

```
    ...
   </S:Detail>
  </S:Fault>
 </S:Body>
</S:Envelope>
```

The properties above bind to a SOAP 1.1 fault as follows when the fault is triggered by processing an RM header block:

```
<S11:Envelope>
 <S11:Header>
   <wsrm:SequenceFault>
     <wsrm:FaultCode> wsrm:FaultCodes </wsrm:FaultCode>
     <wsrm:Detail> [Detail] </wsrm:Detail>
     ...
   </wsrm:SequenceFault>
   <!-- Headers elided for brevity.  -->
 </S11:Header>
 <S11:Body>
  <S11:Fault>
   <faultcode> [Code] </faultcode>
   <faultstring> [Reason] </faultstring>
  </S11:Fault>
 </S11:Body>
</S11:Envelope>
```

The properties bind to a SOAP 1.1 fault as follows when the fault is generated as a result of processing a `CreateSequence` request message:

```
<S11:Envelope>
 <S11:Body>
  <S11:Fault>
   <faultcode> [Subcode] </faultcode>
   <faultstring> [Reason] </faultstring>
  </S11:Fault>
 </S11:Body>
</S11:Envelope>
```

4.1 SequenceFault Element

The purpose of the `SequenceFault` element is to carry the specific details of a fault generated during the reliable messaging specific processing of a message belonging to a Sequence. WS-ReliableMessaging nodes MUST use the `SequenceFault` container only in conjunction with the SOAP 1.1 fault mechanism. WS-ReliableMessaging nodes MUST NOT use the `SequenceFault` container in conjunction with the SOAP 1.2 binding.

The following exemplar defines its syntax:

```
<wsrm:SequenceFault ...>
  <wsrm:FaultCode> wsrm:FaultCode </wsrm:FaultCode>
  <wsrm:Detail> ... </wsrm:Detail> ?
  ...
</wsrm:SequenceFault>
```

The following describes the content model of the `SequenceFault` element.

/wsrm:SequenceFault

This is the element containing Sequence fault information for WS-ReliableMessaging

/wsrm:SequenceFault/wsrm:FaultCode

WS-ReliableMessaging nodes that generate a `SequenceFault` MUST set the value of this element to a qualified name from the set of faults [Subcodes] defined below.

`/wsrm:SequenceFault/wsrm:Detail`

This element, if present, carries application specific error information related to the fault being described.

`/wsrm:SequenceFault/wsrm:Detail/{any}`

The application specific error information related to the fault being described.

`/wsrm:SequenceFault/wsrm:Detail/@{any}`

The application specific error information related to the fault being described.

`/wsrm:SequenceFault/{any}`

This is an extensibility mechanism to allow different (extensible) types of information, based on a schema, to be passed.

`/wsrm:SequenceFault/@{any}`

This is an extensibility mechanism to allow additional attributes, based on schemas, to be added to the element.

4.2 Sequence Terminated

The Endpoint that generates this fault SHOULD make every reasonable effort to notify the corresponding Endpoint of this decision.

Properties:

[Code] Sender or Receiver

[Subcode] wsrm:SequenceTerminated

[Reason] The Sequence has been terminated due to an unrecoverable error.

[Detail]

```
<wsrm:Identifier ...> xs:anyURI </wsrm:Identifier>
```

Generated by	Condition	Action Upon Generation	Action Upon Receipt
RM Source or RM Destination.	Encountering an unrecoverable condition or detection of violation of the protocol.	Sequence termination.	MUST terminate the Sequence if not otherwise terminated.

4.3 Unknown Sequence

Properties:

[Code] Sender

[Subcode] wsrm:UnknownSequence

[Reason] The value of `wsrm:Identifier` is not a known Sequence identifier.

[Detail]

```
<wsrm:Identifier ...> xs:anyURI </wsrm:Identifier>
```

Generated by	Condition	Action Upon Generation	Action Upon Receipt
RM Source or RM Destination.	In response to a message containing an unknown or terminated Sequence identifier.	None.	MUST terminate the Sequence if not otherwise terminated.

4.4 Invalid Acknowledgement

An example of when this fault is generated is when a message is Received by the RM Source containing a `SequenceAcknowledgement` covering messages that have not been sent.

[Code] Sender

[Subcode] wsrm:InvalidAcknowledgement

[Reason] The `SequenceAcknowledgement` violates the cumulative Acknowledgement invariant.

[Detail]

```
<wsrm:SequenceAcknowledgement ...> ... </wsrm:SequenceAcknowledgement>
```

Generated by	Condition	Action Upon Generation	Action Upon Receipt
RM Source.	In response to a `SequenceAcknowledgement` that violate the invariants stated in 2.3 or any of the requirements in 3.9 about valid combinations of AckRange, `Nack` and `None` in a single `SequenceAcknowledgement` element or with respect to already Received such elements.	Unspecified.	Unspecified.

4.5 Message Number Rollover

If the condition listed below is reached, the RM Destination MUST generate this fault.

Properties:

[Code] Sender

[Subcode] wsrm:MessageNumberRollover

[Reason] The maximum value for `wsrm:MessageNumber` has been exceeded.

[Detail]

```
<wsrm:Identifier ...> xs:anyURI </wsrm:Identifier>
<wsrm:MaxMessageNumber> wsrm:MessageNumberType </wsrm:MaxMessageNumber>
```

Generated by	Condition	Action Upon Generation	Action Upon Receipt
RM Destination.	Message number in `/wsrm:Sequence/wsrm:MessageNumber` of a Received message exceeds the internal limitations of an RM Destination or reaches the maximum value of 9,223,372,036,854,775,807.	RM Destination SHOULD continue to accept undelivered messages until the Sequence is closed or terminated.	RM Source SHOULD continue to retransmit undelivered messages until the Sequence is closed or terminated.

4.6 Create Sequence Refused

Properties:

[Code] Sender or Receiver

[Subcode] wsrm:CreateSequenceRefused

[Reason] The Create Sequence request has been refused by the RM Destination.

[Detail]

```
xs:any
```

Generated by	Condition	Action Upon Generation	Action Upon Receipt
RM Destination.	In response to a `CreateSequence` message when the RM Destination does not wish to create a new Sequence.	Unspecified.	Sequence terminated.

4.7 Sequence Closed

This fault is generated by an RM Destination to indicate that the specified Sequence has been closed. This fault MUST be generated when an RM Destination is asked to accept a message for a Sequence that is closed.

Properties:

[Code] Sender

[Subcode] wsrm:SequenceClosed

[Reason] The Sequence is closed and cannot accept new messages.

[Detail]

```
<wsrm:Identifier...> xs:anyURI </wsrm:Identifier>
```

Generated by	Condition	Action Upon Generation	Action Upon Receipt
RM Destination.	In response to a message that belongs to a Sequence that is already closed.	Unspecified.	Sequence closed.

4.8 WSRM Required

If an RM Destination requires the use of WS-RM, this fault is generated when it Receives an incoming message that did not use this protocol.

Properties:

[Code] Sender

[Subcode] wsrm:WSRMRequired

[Reason] The RM Destination requires the use of WSRM.

[Detail]

```
xs:any
```

5 Security Threats and Countermeasures

This specification considers two sets of security requirements, those of the applications that use the WS-RM protocol and those of the protocol itself.

This specification makes no assumptions about the security requirements of the applications that use WS-RM. However, once those requirements have been satisfied within a given operational context, the addition of WS-RM to this operational context should not undermine the fulfillment of those requirements; the use of WS-RM should not create additional attack vectors within an otherwise secure system.

There are many other security concerns that one may need to consider when implementing or using this protocol. The material below should not be considered as a "check list". Implementers and users of this protocol are urged to perform a security analysis to determine their particular threat profile and the appropriate responses to those threats.

Implementers are also advised that there is a core tension between security and reliable messaging that can be problematic if not addressed by implementations; one aspect of security is to prevent message replay but one of the invariants of this protocol is to resend messages until they are acknowledged. Consequently, if the security sub-system processes a message but a failure occurs before the reliable messaging sub-system Receives that message, then it is possible (and likely) that the security sub-system will treat subsequent copies as replays and discard them. At the same time, the reliable messaging sub-system will likely continue to expect and even solicit the missing message(s). Care should be taken to avoid and prevent this condition.

5.1 Threats and Countermeasures

The primary security requirement of this protocol is to protect the specified semantics and protocol invariants against various threats. The following sections describe several threats to the integrity and operation of this protocol and provide some general outlines of countermeasures to those threats. Implementers and users of this protocol should keep in mind that all threats are not necessarily applicable to all operational contexts.

5.1.1 Integrity Threats

In general, any mechanism which allows an attacker to alter the information in a Sequence Traffic Message, Sequence Lifecycle Message, Acknowledgement Messages, Acknowledgement Request, or Sequence-related fault, or which allows an attacker to alter the correlation of a RM Protocol Header Block to its intended message represents a threat to the WS-RM protocol.

For example, if an attacker is able to swap `Sequence` headers on messages in transit between the RM Source and RM Destination then they have undermined the implementation's ability to guarantee the first invariant described in section 2.3. The result is that there is no way of guaranteeing that messages will be Delivered to the Application Destination in the same order that they were sent by the Application Source.

5.1.1.1 Countermeasures

Integrity threats are generally countered via the use of digital signatures some level of the communication protocol stack. Note that, in order to counter header swapping attacks, the signature SHOULD include both the SOAP body and any relevant SOAP headers (e.g. `Sequence` header). Because some headers (`AckRequested`, `SequenceAcknowledgement`) are independent of the body of the SOAP message in which they occur, implementations MUST allow for signatures that cover only these headers.

5.1.2 Resource Consumption Threats

The creation of a Sequence with an RM Destination consumes various resources on the systems used to implement that RM Destination. These resources can include network connections, database tables, message queues, etc. This behavior can be exploited to conduct denial of service attacks against an RM Destination. For example, a simple attack is to repeatedly send `CreateSequence` messages to an RM Destination. Another attack is to create a Sequence for a service that is known to require in-order message Delivery and use this Sequence to send a stream of very large messages to that service, making sure to omit message number “1” from that stream.

5.1.2.1 Countermeasures

There are a number of countermeasures against the described resource consumption threats. The technique advocated by this specification is for the RM Destination to restrict the ability to create a Sequence to a specific set of entities/principals. This reduces the number of potential attackers and, in some cases, allows the identity of any attackers to be determined.

The ability to restrict Sequence creation depends, in turn, upon the RM Destination's ability to identify and authenticate the RM Source that issued the `CreateSequence` message.

5.1.3 Sequence Spoofing Threats

Sequence spoofing is a class of threats in which the attacker uses knowledge of the `Identifier` for a particular Sequence to forge Sequence Lifecycle or Traffic Messages. For example the attacker creates a fake `TerminateSequence` message that references the target Sequence and sends this message to the appropriate RM Destination. Some Sequence spoofing attacks also require up-to-date knowledge of the current `MessageNumber` for their target Sequence.

In general any Sequence Lifecycle Message, RM Protocol Header Block, or Sequence-correlated SOAP fault (e.g. `InvalidAcknowledgement`) can be used by someone with knowledge of the Sequence `Identifier` to attack the Sequence. These attacks are “two-way” in that an attacker may choose to target the RM Source by, for example, inserting a fake `SequenceAcknowledgement` header into a message that it sends to the `AcksTo` EPR of an RM Source.

5.1.3.1 Sequence Hijacking

Sequence hijacking is a specific case of a Sequence spoofing attack. The attacker attempts to inject Sequence Traffic Messages into an existing Sequence by inserting fake `Sequence` headers into those messages.

Note that “Sequence hijacking” should not be equated with “security session hijacking”. Although a Sequence may be bound to some form of a security session in order to counter the threats described in this section, applications MUST NOT rely on WS-RM-related information to make determinations about the identity of the entity that created a message; applications SHOULD rely only upon information that is established by the security infrastructure to make such determinations. Failure to observe this rule creates, among other problems, a situation in which the absence of WS-RM may deprive an application of the ability to authenticate its peers even though the necessary security processing has taken place.

5.1.3.2 Countermeasures

There are a number of countermeasures against Sequence spoofing threats. The technique advocated by this specification is to consider the Sequence to be a shared resource that is jointly owned by the RM Source that initiated its creation (i.e. that sent the `CreateSequence` message) and the RM Destination that serves as its terminus (i.e. that sent the `CreateSequenceResponse` message). To counter

Sequence spoofing attempts the RM Destination SHOULD ensure that every message or fault that it Receives that refers to a particular Sequence originated from the RM Source that jointly owns the referenced Sequence. For its part the RM Source SHOULD ensure that every message or fault that it Receives that refers to a particular Sequence originated from the RM Destination that jointly owns the referenced Sequence.

For the RM Destination to be able to identify its Sequence peer it MUST be able to identify and authenticate the entity that sent the `CreateSequence` message. Similarly for the RM Source to identify its Sequence peer it MUST be able to identify and authenticate the entity that sent the `CreateSequenceResponse` message. For either the RM Destination or the RM Source to determine if a message was sent by its Sequence peer it MUST be able to identify and authenticate the initiator of that message and, if necessary, correlate this identity with the Sequence peer identity established at Sequence creation time.

5.2 Security Solutions and Technologies

The security threats described in the previous sections are neither new nor unique. The solutions that have been developed to secure other SOAP-based protocols can be used to secure WS-RM as well. This section maps the facilities provided by common web services security solutions against countermeasures described in the previous sections.

Before continuing this discussion, however, some examination of the underlying requirements of the previously described countermeasures is necessary. Specifically it should be noted that the technique described in section 5.1.2.1 has two components. Firstly, the RM Destination identifies and authenticates the issuer of a `CreateSequence` message. Secondly, the RM Destination performs an authorization check against this authenticated identity and determines if the RM Source is permitted to create Sequences with the RM Destination. Since the facilities for performing this authorization check (runtime infrastructure, policy frameworks, etc.) lie completely within the domain of individual implementations, any discussion of such facilities is considered to be beyond the scope of this specification.

5.2.1 Transport Layer Security

This section describes how the facilities provided by SSL/TLS [RFC 4346] can be used to implement the countermeasures described in the previous sections. The use of SSL/TLS is subject to the constraints defined in section 4 of the Basic Security Profile 1.0 [BSP 1.0].

The description provided here is general in nature and is not intended to serve as a complete definition on the use of SSL/TLS to protect WS-RM. In order to interoperate implementations need to agree on the choice of features as well as the manner in which they will be used. The mechanisms described in the Web Services Security Policy Language [SecurityPolicy] MAY be used by services to describe the requirements and constraints of the use of SSL/TLS.

5.2.1.1 Model

The basic model for using SSL/TLS is as follows:

1. The RM Source establishes an SSL/TLS session with the RM Destination.
2. The RM Source uses this SSL/TLS session to send a `CreateSequence` message to the RM Destination.
3. The RM Destination establishes an SSL/TLS session with the RM Source and sends an asynchronous `CreateSequenceResponse` using this session. Alternately it may respond with a synchronous `CreateSequenceResponse` using the session established in (1).

4. For the lifetime of the Sequence the RM Source uses the SSL/TLS session from (1) to Transmit any and all messages or faults that refer to that Sequence.
5. For the lifetime of the Sequence the RM Destination either uses the SSL/TLS session established in (3) to Transmit any and all messages or faults that refer to that Sequence or, for synchronous exchanges, the RM Destination uses the SSL/TLS session established in (1).

5.2.1.2 Countermeasure Implementation

Used in its simplest fashion (without relying upon any authentication mechanisms), SSL/TLS provides the necessary integrity qualities to counter the threats described in section 5.1.1. Note, however, that the nature of SSL/TLS limits the scope of this integrity protection to a single transport level session. If SSL/TLS is the only mechanism used to provide integrity, any intermediaries between the RM Source and the RM Destination MUST be trusted to preserve the integrity of the messages that flow through them.

As noted, the technique described in sections 5.1.2.1 involves the use of authentication. This specification advocates either of two mechanisms for authenticating entities using SSL/TLS. In both of these methods the SSL/TLS server (the party accepting the SSL/TLS connection) authenticates itself to the SSL/TLS client using an X.509 certificate that is exchanged during the SSL/TLS handshake.

- **HTTP Basic Authentication**: This method of authentication presupposes that a SOAP/HTTP binding is being used as part of the protocol stack beneath WS-RM. Subsequent to the establishment of the SSL/TLS session, the sending party authenticates itself to the receiving party using HTTP Basic Authentication [RFC 2617]. For example, a RM Source might authenticate itself to a RM Destination (e.g. when transmitting a Sequence Traffic Message) using BasicAuth. Similarly the RM Destination might authenticate itself to the RM Source (e.g. when sending an Acknowledgement) using BasicAuth.
- **SSL/TLS Client Authentication:** In this method of authentication, the party initiating the connection authenticates itself to the party accepting the connection using an X.509 certificate that is exchanged during the SSL/TLS handshake.

To implement the countermeasures described in section 5.1.2.1 the RM Source must authenticate itself using one the above mechanisms. The authenticated identity can then be used to determine if the RM Source is authorized to create a Sequence with the RM Destination.

This specification advocates implementing the countermeasures described in section 5.1.3.2 by requiring an RM node's Sequence peer to be equivalent to their SSL/TLS session peer. This allows the authorization decisions described in section 5.1.3.2 to be based on SSL/TLS session identity rather than on authentication information. For example, an RM Destination can determine that a Sequence Traffic Message rightfully belongs to its referenced Sequence if that message arrived over the same SSL/TLS session that was used to carry the `CreateSequence` message for that Sequence. Note that requiring a one-to-one relationship between SSL/TLS session peer and Sequence peer constrains the lifetime of a SSL/TLS-protected Sequence to be less than or equal to the lifetime of the SSL/TLS session that is used to protect that Sequence.

This specification does not preclude the use of other methods of using SSL/TLS to implement the countermeasures (such as associating specific authentication information with a Sequence) although such methods are not covered by this document.

Issues specific to the life-cycle management of SSL/TLS sessions (such as the resumption of a SSL/TLS session) are outside the scope of this specification.

5.2.2 SOAP Message Security

The mechanisms described in WS-Security may be used in various ways to implement the countermeasures described in the previous sections. This specification advocates using the protocol described by WS-SecureConversation [SecureConversation] (optionally in conjunction with WS-Trust

[Trust]) as a mechanism for protecting Sequences. The use of WS-Security (as an underlying component of WS-SecureConversation) is subject to the constraints defined in the Basic Security Profile 1.0.

The description provided here is general in nature and is not intended to serve as a complete definition on the use of WS-SecureConversation/WS-Trust to protect WS-RM. In order to interoperate implementations need to agree on the choice of features as well as the manner in which they will be used. The mechanisms described in the Web Services Security Policy Language MAY be used by services to describe the requirements and constraints of the use of WS-SecureConversation.

5.2.2.1 Model

The basic model for using WS-SecureConversation is as follows:

1 The RM Source and the RM Destination create a WS-SecureConversation security context. This may involve the participation of third parties such as a security token service. The tokens exchanged may contain authentication claims (e.g. X.509 certificates or Kerberos service tickets).

2 During the `CreateSequence` exchange, the RM Source SHOULD explicitly identify the security context that will be used to protect the Sequence. This is done so that, in cases where the `CreateSequence` message is signed by more than one security context, the RM Source can indicate which security context should be used to protect the newly created Sequence.

3 For the lifetime of the Sequence the RM Source and the RM Destination use the session key(s) associated with the security context to sign (as defined by WS-Security) at least the body and any relevant WS-RM-defined headers of any and all messages or faults that refer to that Sequence.

5.2.2.2 Countermeasure Implementation

Without relying upon any authentication information, the per-message signatures provide the necessary integrity qualities to counter the threats described in section 5.1.1.

To implement the countermeasures described in section 5.1.2.1 some mutually agreed upon form of authentication claims must be provided by the RM Source to the RM Destination during the establishment of the Security Context. These claims can then be used to determine if the RM Source is authorized to create a Sequence with the RM Destination.

This specification advocates implementing the countermeasures described in section 5.1.3.2 by requiring an RM node's Sequence peer to be equivalent to their security context session peer. This allows the authorization decisions described in section 5.1.3.2 to be based on the identity of the message's security context rather than on any authentication claims that may have been established during security context initiation. Note that other methods of using WS-SecureConversation to implement the countermeasures (such as associating specific authentication claims to a Sequence) are possible but not covered by this document.

As with transport security, the requisite equivalence of a security context peer with a Sequence peer limits the lifetime of a Sequence to the lifetime of the protecting security context. Unlike transport security, the association between a Sequence and its protecting security context cannot always be established implicitly at Sequence creation time. This is due to the fact that the `CreateSequence` and `CreateSequenceResponse` messages may be signed by more than one security context.

Issues specific to the life-cycle management of WS-SecureConversation security contexts (such as amending or renewing contexts) are outside the scope of this specification.

6 Securing Sequences

As noted in section 5, the RM Source and RM Destination should be able to protect their shared Sequences against the threat of Sequence Spoofing attacks. There are a number of OPTIONAL means of achieving this objective depending upon the underlying security infrastructure.

6.1 Securing Sequences Using WS-Security

One mechanism for protecting a Sequence is to include a security token using a `wsse:SecurityTokenReference` element from WS-Security (see section 9 in WS-SecureConversation) in the `CreateSequence` element. This establishes an association between the created (and, if present, offered) Sequence(s) and the referenced security token, such that the RM Source and Destination MUST use the security token as the basis for authorization of all subsequent interactions related to the Sequence(s). The `wsse:SecurityTokenReference` explicitly identifies the token as there may be more than one token on a `CreateSequence` message or inferred from the communication context (e.g. transport protection).

It is RECOMMENDED that a message independent referencing mechanism be used to identify the token, if the token being referenced supports such mechanism.

The following exemplar defines the `CreateSequence` syntax when extended to include a `wsse:SecurityTokenReference`:

```
<wsrm:CreateSequence ...>
    <wsrm:AcksTo> wsa:EndpointReferenceType </wsrm:AcksTo>
    <wsrm:Expires ...> xs:duration </wsrm:Expires> ?
    <wsrm:Offer ...>
        <wsrm:Identifier ...> xs:anyURI </wsrm:Identifier>
        <wsrm:Endpoint> wsa:EndpointReferenceType </wsrm:Endpoint>
        <wsrm:Expires ...> xs:duration </wsrm:Expires> ?
        <wsrm:IncompleteSequenceBehavior>
            wsrm:IncompleteSequenceBehaviorType
        </wsrm:IncompleteSequenceBehavior> ?
        ...
    </wsrm:Offer> ?
    ...
    <wsse:SecurityTokenReference>
      ...
    </wsse:SecurityTokenReference> ?
    ...
</wsrm:CreateSequence>
```

The following describes the content model of the additional `CreateSequence` elements.

`/wsrm:CreateSequence/wsse:SecurityTokenReference`

> This element uses the extensibility mechanism defined for the `CreateSequence` element (defined in section 3.4) to communicate an explicit reference to the security token, using a `wsse:SecurityTokenReference` as documented in WS-Security, that the RM Source and Destination MUST use to authorize messages for the created (and, if present, the offered) Sequence(s). All subsequent messages related to the created (and, if present, the offered) Sequence(s) MUST demonstrate proof-of-possession of the secret associated with the token (e.g., by using or deriving from a private or secret key).

When a RM Source transmits a `CreateSequence` that has been extended to include a `wsse:SecurityTokenReference` it SHOULD ensure that the RM Destination both understands and will conform to the requirements listed above. In order to achieve this, the RM Source SHOULD include the `UsesSequenceSTR` element as a SOAP header block within the `CreateSequence` message. This element MUST include a `soap:mustUnderstand` attribute with a value of ‘true’. Thus the RM Source can be assured that a RM Destination that responds with a `CreateSequenceResponse` understands and conforms with the requirements listed above. Note that an RM Destination understanding this header does not mean that it has processed and understood any WS-Security headers, the fault behavior defined in WS-Security still applies.

The following exemplar defines the `UsesSequenceSTR` syntax:

```
<wsrm:UsesSequenceSTR ... />
```

The following describes the content model of the `UsesSequenceSTR` header block.

`/wsrm`:UsesSequenceSTR

> This element SHOULD be included as a SOAP header block in `CreateSequence` messages that use the extensibility mechanism described above in this section. The `soap:mustUnderstand` attribute value MUST be ‘true’. The receiving RM Destination MUST understand and correctly implement the extension described above or else generate a `soap:MustUnderstand` fault, thus aborting the requested Sequence creation.

The following is an example of a `CreateSequence` message using the `wsse:SecurityTokenReference` extension and the `UsesSequenceSTR` header block:

```
<soap:Envelope ...>
  <soap:Header>
    ...
    <wsrm:UsesSequenceSTR soap:mustUnderstand='true'/>
    ...
  </soap:Header>
  <soap:Body>
    <wsrm:CreateSequence>
      <wsrm:AcksTo>
        <wsa:Address>http://Business456.com/serviceA/789</wsa:Address>
      </wsrm:AcksTo>
      <wsse:SecurityTokenReference>
        ...
      </wsse:SecurityTokenReference>
    </wsrm:CreateSequence>
  </soap:Body>
</soap:Envelope>
```

6.2 Securing Sequences Using SSL/TLS

One mechanism for protecting a Sequence is to bind the Sequence to the underlying SSL/TLS session(s). The RM Source indicates to the RM Destination that a Sequence is to be bound to the underlying SSL/TLS session(s) via the `UsesSequenceSSL` header block. If the RM Source wishes to bind a Sequence to the underlying SSL/TLS sessions(s) it MUST include the `UsesSequenceSSL` element as a SOAP header block within the `CreateSequence` message.

The following exemplar defines the `UsesSequenceSSL` syntax:

```
<wsrm:UsesSequenceSSL soap:mustUnderstand="true" ... />
```

The following describes the content model of the `UsesSequenceSSL` header block.

`/wsrm`:UsesSequenceSSL

> The RM Source MAY include this element as a SOAP header block of a `CreateSequence` message to indicate to the RM Destination that the resulting Sequence is to be bound to the

SSL/TLS session that was used to carry the `CreateSequence` message. If included, the RM Source MUST mark this header with a `soap:mustUnderstand` attribute with a value of ‘true’. The receiving RM Destination MUST understand and correctly implement the functionality described in section 5.2.1 or else generate a `soap:MustUnderstand` fault, thus aborting the requested Sequence creation.

Note that the inclusion of the above header by the RM Source implies that all Sequence-related information (Sequence Lifecycle or Acknowledgment messages or Sequence-related faults) flowing from the RM Destination to the RM Source will be bound to the SSL/TLS session that is used to carry the `CreateSequenceResponse` message.

Appendix A. Schema

The normative schema that is defined for WS-ReliableMessaging using [XML-Schema Part1] and [XML-Schema Part2] is located at:

```
http://docs.oasis-open.org/ws-rx/wsrm/200702/wsrm-1.1-schema-200702.xsd
```

The following copy is provided for reference.

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Copyright(C) OASIS(R) 1993-2007. All Rights Reserved.
     OASIS trademark, IPR and other policies apply.  -->
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:wsa="http://www.w3.org/2005/08/addressing" xmlns:wsrm="http://docs.oasis-
open.org/ws-rx/wsrm/200702" targetNamespace="http://docs.oasis-open.org/ws-
rx/wsrm/200702" elementFormDefault="qualified"
attributeFormDefault="unqualified">
  <xs:import namespace="http://www.w3.org/2005/08/addressing"
schemaLocation="http://www.w3.org/2006/03/addressing/ws-addr.xsd"/>
  <!-- Protocol Elements -->
  <xs:complexType name="SequenceType">
    <xs:sequence>
      <xs:element ref="wsrm:Identifier"/>
      <xs:element name="MessageNumber" type="wsrm:MessageNumberType"/>
      <xs:any namespace="##other" processContents="lax" minOccurs="0"
maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:anyAttribute namespace="##other" processContents="lax"/>
  </xs:complexType>
  <xs:element name="Sequence" type="wsrm:SequenceType"/>
  <xs:element name="SequenceAcknowledgement">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="wsrm:Identifier"/>
        <xs:choice>
          <xs:sequence>
            <xs:choice>
              <xs:element name="AcknowledgementRange" maxOccurs="unbounded">
                <xs:complexType>
                  <xs:sequence/>
                  <xs:attribute name="Upper" type="xs:unsignedLong"
use="required"/>
                  <xs:attribute name="Lower" type="xs:unsignedLong"
use="required"/>
                  <xs:anyAttribute namespace="##other" processContents="lax"/>
                </xs:complexType>
              </xs:element>
              <xs:element name="None">
                <xs:complexType>
                  <xs:sequence/>
                </xs:complexType>
              </xs:element>
            </xs:choice>
            <xs:element name="Final" minOccurs="0">
              <xs:complexType>
                <xs:sequence/>
              </xs:complexType>
            </xs:element>
          </xs:sequence>
          <xs:element name="Nack" type="xs:unsignedLong"
```

```
maxOccurs="unbounded"/>
        </xs:choice>
        <xs:any namespace="##other" processContents="lax" minOccurs="0"
maxOccurs="unbounded"/>
      </xs:sequence>
      <xs:anyAttribute namespace="##other" processContents="lax"/>
    </xs:complexType>
  </xs:element>
  <xs:complexType name="AckRequestedType">
    <xs:sequence>
      <xs:element ref="wsrm:Identifier"/>
      <xs:any namespace="##other" processContents="lax" minOccurs="0"
maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:anyAttribute namespace="##other" processContents="lax"/>
  </xs:complexType>
  <xs:element name="AckRequested" type="wsrm:AckRequestedType"/>
  <xs:element name="Identifier">
    <xs:complexType>
      <xs:annotation>
        <xs:documentation>
          This type is for elements whose [children] is an anyURI and can have
arbitrary attributes.
        </xs:documentation>
      </xs:annotation>
      <xs:simpleContent>
        <xs:extension base="xs:anyURI">
          <xs:anyAttribute namespace="##other" processContents="lax"/>
        </xs:extension>
      </xs:simpleContent>
    </xs:complexType>
  </xs:element>
  <xs:element name="Address">
    <xs:complexType>
      <xs:simpleContent>
        <xs:extension base="xs:anyURI">
          <xs:anyAttribute namespace="##other" processContents="lax"/>
        </xs:extension>
      </xs:simpleContent>
    </xs:complexType>
  </xs:element>
  <xs:simpleType name="MessageNumberType">
    <xs:restriction base="xs:unsignedLong">
      <xs:minInclusive value="1"/>
      <xs:maxInclusive value="9223372036854775807"/>
    </xs:restriction>
  </xs:simpleType>
  <!-- Fault Container and Codes -->
  <xs:simpleType name="FaultCodes">
    <xs:restriction base="xs:QName">
      <xs:enumeration value="wsrm:SequenceTerminated"/>
      <xs:enumeration value="wsrm:UnknownSequence"/>
      <xs:enumeration value="wsrm:InvalidAcknowledgement"/>
      <xs:enumeration value="wsrm:MessageNumberRollover"/>
      <xs:enumeration value="wsrm:CreateSequenceRefused"/>
      <xs:enumeration value="wsrm:SequenceClosed"/>
      <xs:enumeration value="wsrm:WSRMRequired"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:complexType name="SequenceFaultType">
    <xs:sequence>
      <xs:element name="FaultCode" type="wsrm:FaultCodes"/>
      <xs:element name="Detail" type="wsrm:DetailType" minOccurs="0"/>
      <xs:any namespace="##other" processContents="lax" minOccurs="0"
```

```
maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:anyAttribute namespace="##other" processContents="lax"/>
  </xs:complexType>
  <xs:complexType name="DetailType">
    <xs:sequence>
      <xs:any namespace="##other" processContents="lax" minOccurs="0"
maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:anyAttribute namespace="##other" processContents="lax"/>
  </xs:complexType>
  <xs:element name="SequenceFault" type="wsrm:SequenceFaultType"/>
  <xs:element name="CreateSequence" type="wsrm:CreateSequenceType"/>
  <xs:element name="CreateSequenceResponse"
type="wsrm:CreateSequenceResponseType"/>
  <xs:element name="CloseSequence" type="wsrm:CloseSequenceType"/>
  <xs:element name="CloseSequenceResponse"
type="wsrm:CloseSequenceResponseType"/>
  <xs:element name="TerminateSequence" type="wsrm:TerminateSequenceType"/>
  <xs:element name="TerminateSequenceResponse"
type="wsrm:TerminateSequenceResponseType"/>
  <xs:complexType name="CreateSequenceType">
    <xs:sequence>
      <xs:element ref="wsrm:AcksTo"/>
      <xs:element ref="wsrm:Expires" minOccurs="0"/>
      <xs:element name="Offer" type="wsrm:OfferType" minOccurs="0"/>
      <xs:any namespace="##other" processContents="lax" minOccurs="0"
maxOccurs="unbounded">
        <xs:annotation>
          <xs:documentation>
            It is the authors intent that this extensibility be used to
transfer a Security Token Reference as defined in WS-Security.
          </xs:documentation>
        </xs:annotation>
      </xs:any>
    </xs:sequence>
    <xs:anyAttribute namespace="##other" processContents="lax"/>
  </xs:complexType>
  <xs:complexType name="CreateSequenceResponseType">
    <xs:sequence>
      <xs:element ref="wsrm:Identifier"/>
      <xs:element ref="wsrm:Expires" minOccurs="0"/>
      <xs:element name="IncompleteSequenceBehavior"
type="wsrm:IncompleteSequenceBehaviorType" minOccurs="0"/>
      <xs:element name="Accept" type="wsrm:AcceptType" minOccurs="0"/>
      <xs:any namespace="##other" processContents="lax" minOccurs="0"
maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:anyAttribute namespace="##other" processContents="lax"/>
  </xs:complexType>
  <xs:complexType name="CloseSequenceType">
    <xs:sequence>
      <xs:element ref="wsrm:Identifier"/>
      <xs:element name="LastMsgNumber" type="wsrm:MessageNumberType"
minOccurs="0"/>
      <xs:any namespace="##other" processContents="lax" minOccurs="0"
maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:anyAttribute namespace="##other" processContents="lax"/>
  </xs:complexType>
  <xs:complexType name="CloseSequenceResponseType">
    <xs:sequence>
      <xs:element ref="wsrm:Identifier"/>
      <xs:any namespace="##other" processContents="lax" minOccurs="0"
```

```
maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:anyAttribute namespace="##other" processContents="lax"/>
  </xs:complexType>
  <xs:complexType name="TerminateSequenceType">
    <xs:sequence>
      <xs:element ref="wsrm:Identifier"/>
      <xs:element name="LastMsgNumber" type="wsrm:MessageNumberType"
minOccurs="0"/>
      <xs:any namespace="##other" processContents="lax" minOccurs="0"
maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:anyAttribute namespace="##other" processContents="lax"/>
  </xs:complexType>
  <xs:complexType name="TerminateSequenceResponseType">
    <xs:sequence>
      <xs:element ref="wsrm:Identifier"/>
      <xs:any namespace="##other" processContents="lax" minOccurs="0"
maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:anyAttribute namespace="##other" processContents="lax"/>
  </xs:complexType>
  <xs:element name="AcksTo" type="wsa:EndpointReferenceType"/>
  <xs:complexType name="OfferType">
    <xs:sequence>
      <xs:element ref="wsrm:Identifier"/>
      <xs:element name="Endpoint" type="wsa:EndpointReferenceType"/>
      <xs:element ref="wsrm:Expires" minOccurs="0"/>
      <xs:element name="IncompleteSequenceBehavior"
type="wsrm:IncompleteSequenceBehaviorType" minOccurs="0"/>
      <xs:any namespace="##other" processContents="lax" minOccurs="0"
maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:anyAttribute namespace="##other" processContents="lax"/>
  </xs:complexType>
  <xs:complexType name="AcceptType">
    <xs:sequence>
      <xs:element ref="wsrm:AcksTo"/>
      <xs:any namespace="##other" processContents="lax" minOccurs="0"
maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:anyAttribute namespace="##other" processContents="lax"/>
  </xs:complexType>
  <xs:element name="Expires">
    <xs:complexType>
      <xs:simpleContent>
        <xs:extension base="xs:duration">
          <xs:anyAttribute namespace="##other" processContents="lax"/>
        </xs:extension>
      </xs:simpleContent>
    </xs:complexType>
  </xs:element>
  <xs:simpleType name="IncompleteSequenceBehaviorType">
    <xs:restriction base="xs:string">
      <xs:enumeration value="DiscardEntireSequence"/>
      <xs:enumeration value="DiscardFollowingFirstGap"/>
      <xs:enumeration value="NoDiscard"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:element name="UsesSequenceSTR">
    <xs:complexType>
      <xs:sequence/>
      <xs:anyAttribute namespace="##other" processContents="lax"/>
    </xs:complexType>
  </xs:element>
  <xs:element name="UsesSequenceSSL">
    <xs:complexType>
      <xs:sequence/>
      <xs:anyAttribute namespace="##other" processContents="lax"/>
    </xs:complexType>
  </xs:element>
  <xs:element name="UnsupportedElement">
    <xs:simpleType>
      <xs:restriction base="xs:QName"/>
    </xs:simpleType>
  </xs:element>
</xs:schema>
```

Appendix B. WSDL

This WSDL describes the WS-RM protocol from the point of view of an RM Destination. In the case where an endpoint acts both as an RM Destination and an RM Source, note that additional messages may be present in exchanges with that endpoint.

Also note that this WSDL is intended to describe the internal structure of the WS-RM protocol, and will not generally appear in a description of a WS-RM-capable Web service. See WS-RM Policy [WS-RM Policy] for a higher-level mechanism to indicate that WS-RM is engaged.

The normative WSDL 1.1 definition for WS-ReliableMessaging is located at:

```
http://docs.oasis-open.org/ws-rx/wsrm/200702/wsrm-1.1-wsdl-200702e1.wsdl
```

The following non-normative copy is provided for reference.

```
<?xml version="1.0" encoding="utf-8"?>
<!-- Copyright(C) OASIS(R) 1993-2007. All Rights Reserved.
     OASIS trademark, IPR and other policies apply.  -->
<wsdl:definitions xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:wsa="http://www.w3.org/2005/08/addressing"
xmlns:wsam="http://www.w3.org/2007/05/addressing/metadata"
xmlns:rm="http://docs.oasis-open.org/ws-rx/wsrm/200702"
xmlns:tns="http://docs.oasis-open.org/ws-rx/wsrm/200702/wsdl"
targetNamespace="http://docs.oasis-open.org/ws-rx/wsrm/200702/wsdl">

  <wsdl:types>
    <xs:schema>
      <xs:import namespace="http://docs.oasis-open.org/ws-rx/wsrm/200702"
schemaLocation="http://docs.oasis-open.org/ws-rx/wsrm/200702/wsrm-1.1-schema-
200702.xsd"/>
    </xs:schema>
  </wsdl:types>

  <wsdl:message name="CreateSequence">
    <wsdl:part name="create" element="rm:CreateSequence"/>
  </wsdl:message>
  <wsdl:message name="CreateSequenceResponse">
    <wsdl:part name="createResponse" element="rm:CreateSequenceResponse"/>
  </wsdl:message>
  <wsdl:message name="CloseSequence">
    <wsdl:part name="close" element="rm:CloseSequence"/>
  </wsdl:message>
  <wsdl:message name="CloseSequenceResponse">
    <wsdl:part name="closeResponse" element="rm:CloseSequenceResponse"/>
  </wsdl:message>
  <wsdl:message name="TerminateSequence">
    <wsdl:part name="terminate" element="rm:TerminateSequence"/>
  </wsdl:message>
  <wsdl:message name="TerminateSequenceResponse">
    <wsdl:part name="terminateResponse"
element="rm:TerminateSequenceResponse"/>
  </wsdl:message>

  <wsdl:portType name="SequenceAbstractPortType">
    <wsdl:operation name="CreateSequence">
      <wsdl:input message="tns:CreateSequence" wsam:Action="http://docs.oasis-
open.org/ws-rx/wsrm/200702/CreateSequence"/>
      <wsdl:output message="tns:CreateSequenceResponse"
```

```
wsam:Action="http://docs.oasis-open.org/ws-
rx/wsrm/200702/CreateSequenceResponse"/>
    </wsdl:operation>
    <wsdl:operation name="CloseSequence">
      <wsdl:input message="tns:CloseSequence" wsam:Action="http://docs.oasis-
open.org/ws-rx/wsrm/200702/CloseSequence"/>
      <wsdl:output message="tns:CloseSequenceResponse"
wsam:Action="http://docs.oasis-open.org/ws-
rx/wsrm/200702/CloseSequenceResponse"/>
    </wsdl:operation>
    <wsdl:operation name="TerminateSequence">
      <wsdl:input message="tns:TerminateSequence"
wsam:Action="http://docs.oasis-open.org/ws-rx/wsrm/200702/TerminateSequence"/>
      <wsdl:output message="tns:TerminateSequenceResponse"
wsam:Action="http://docs.oasis-open.org/ws-
rx/wsrm/200702/TerminateSequenceResponse"/>
    </wsdl:operation>
  </wsdl:portType>

</wsdl:definitions>
```

Appendix C. Message Examples

Appendix C.1 Create Sequence

Create Sequence

```
<?xml version="1.0" encoding="UTF-8"?>
<S:Envelope xmlns:S="http://www.w3.org/2003/05/soap-envelope"
xmlns:wsrm="http://docs.oasis-open.org/ws-rx/wsrm/200702"
xmlns:wsa="http://www.w3.org/2005/08/addressing">
 <S:Header>
  <wsa:MessageID>
   http://Business456.com/guid/0baaf88d-483b-4ecf-a6d8-a7c2eb546817
  </wsa:MessageID>
  <wsa:To>http://example.com/serviceB/123</wsa:To>
    <wsa:Action>http://docs.oasis-open.org/ws-
rx/wsrm/200702/CreateSequence</wsa:Action>
  <wsa:ReplyTo>
   <wsa:Address>http://Business456.com/serviceA/789</wsa:Address>
  </wsa:ReplyTo>
 </S:Header>
 <S:Body>
  <wsrm:CreateSequence>
    <wsrm:AcksTo>
      <wsa:Address>http://Business456.com/serviceA/789</wsa:Address>
    </wsrm:AcksTo>
  </wsrm:CreateSequence>
 </S:Body>
</S:Envelope>
```

Create Sequence Response

```
<?xml version="1.0" encoding="UTF-8"?>
<S:Envelope xmlns:S="http://www.w3.org/2003/05/soap-envelope"
xmlns:wsrm="http://docs.oasis-open.org/ws-rx/wsrm/200702"
xmlns:wsa="http://www.w3.org/2005/08/addressing">
  <S:Header>
    <wsa:To>http://Business456.com/serviceA/789</wsa:To>
    <wsa:RelatesTo>
      http://Business456.com/guid/0baaf88d-483b-4ecf-a6d8a7c2eb546817
    </wsa:RelatesTo>
    <wsa:Action>
      http://docs.oasis-open.org/ws-rx/wsrm/200702/CreateSequenceResponse
    </wsa:Action>
  </S:Header>
  <S:Body>
    <wsrm:CreateSequenceResponse>
      <wsrm:Identifier>http://Business456.com/RM/ABC</wsrm:Identifier>
    </wsrm:CreateSequenceResponse>
  </S:Body>
</S:Envelope>
```

Appendix C.2 Initial Transmission

The following example WS-ReliableMessaging headers illustrate the message exchange in the above figure. The three messages have the following headers; the third message is identified as the last message in the Sequence:

Message 1

```
<?xml version="1.0" encoding="UTF-8"?>
<S:Envelope xmlns:S="http://www.w3.org/2003/05/soap-envelope"
xmlns:wsrm="http://docs.oasis-open.org/ws-rx/wsrm/200702"
xmlns:wsa="http://www.w3.org/2005/08/addressing">
  <S:Header>
    <wsa:MessageID>
      http://Business456.com/guid/71e0654e-5ce8-477b-bb9d-34f05cfcbc9e
    </wsa:MessageID>
    <wsa:To>http://example.com/serviceB/123</wsa:To>
    <wsa:From>
      <wsa:Address>http://Business456.com/serviceA/789</wsa:Address>
    </wsa:From>
    <wsa:Action>http://example.com/serviceB/123/request</wsa:Action>
    <wsrm:Sequence>
      <wsrm:Identifier>http://Business456.com/RM/ABC</wsrm:Identifier>
      <wsrm:MessageNumber>1</wsrm:MessageNumber>
    </wsrm:Sequence>
  </S:Header>
  <S:Body>
    <!--  Some  Application  Data  -->
  </S:Body>
</S:Envelope>
```

Message 2

```
<?xml version="1.0" encoding="UTF-8"?>
<S:Envelope xmlns:S="http://www.w3.org/2003/05/soap-envelope"
xmlns:wsrm="http://docs.oasis-open.org/ws-rx/wsrm/200702"
xmlns:wsa="http://www.w3.org/2005/08/addressing">
  <S:Header>
    <wsa:MessageID>
      http://Business456.com/guid/daa7d0b2-c8e0-476e-a9a4-d164154e38de
    </wsa:MessageID>
    <wsa:To>http://example.com/serviceB/123</wsa:To>
    <wsa:From>
      <wsa:Address>http://Business456.com/serviceA/789</wsa:Address>
    </wsa:From>
    <wsa:Action>http://example.com/serviceB/123/request</wsa:Action>
    <wsrm:Sequence>
      <wsrm:Identifier>http://Business456.com/RM/ABC</wsrm:Identifier>
      <wsrm:MessageNumber>2</wsrm:MessageNumber>
    </wsrm:Sequence>
  </S:Header>
  <S:Body>
    <!--  Some  Application  Data  -->
  </S:Body>
</S:Envelope>
```

Message 3

```
<?xml version="1.0" encoding="UTF-8"?>
<S:Envelope xmlns:S="http://www.w3.org/2003/05/soap-envelope"
xmlns:wsrm="http://docs.oasis-open.org/ws-rx/wsrm/200702"
xmlns:wsa="http://www.w3.org/2005/08/addressing">
 <S:Header>
  <wsa:MessageID>
   http://Business456.com/guid/0baaf88d-483b-4ecf-a6d8-a7c2eb546819
  </wsa:MessageID>
  <wsa:To>http://example.com/serviceB/123</wsa:To>
  <wsa:From>
   <wsa:Address>http://Business456.com/serviceA/789</wsa:Address>
  </wsa:From>
  <wsa:Action>http://example.com/serviceB/123/request</wsa:Action>
 <wsrm:Sequence>
  <wsrm:Identifier>http://Business456.com/RM/ABC</wsrm:Identifier>
  <wsrm:MessageNumber>3</wsrm:MessageNumber>
 </wsrm:Sequence>
 <wsrm:AckRequested>
  <wsrm:Identifier>http://Business456.com/RM/ABC</wsrm:Identifier>
 </wsrm:AckRequested>
</S:Header>
<S:Body>
 <!-- Some Application Data -->
</S:Body>
</S:Envelope>
```

Appendix C.3 First Acknowledgement

Message number 2 has not been accepted by the RM Destination due to some transmission error so it responds with an Acknowledgement for messages 1 and 3:

```
<?xml version="1.0" encoding="UTF-8"?>
<S:Envelope xmlns:S="http://www.w3.org/2003/05/soap-envelope"
xmlns:wsrm="http://docs.oasis-open.org/ws-rx/wsrm/200702"
xmlns:wsa="http://www.w3.org/2005/08/addressing">
 <S:Header>
  <wsa:MessageID>
   http://example.com/guid/0baaf88d-483b-4ecf-a6d8-a7c2eb546810
  </wsa:MessageID>
  <wsa:To>http://Business456.com/serviceA/789</wsa:To>
  <wsa:From>
   <wsa:Address>http://example.com/serviceB/123</wsa:Address>
  </wsa:From>
  <wsa:Action>
    http://docs.oasis-open.org/ws-rx/wsrm/200702/SequenceAcknowledgement
  </wsa:Action>
  <wsrm:SequenceAcknowledgement>
   <wsrm:Identifier>http://Business456.com/RM/ABC</wsrm:Identifier>
   <wsrm:AcknowledgementRange Upper="1" Lower="1"/>
   <wsrm:AcknowledgementRange Upper="3" Lower="3"/>
  </wsrm:SequenceAcknowledgement>
 </S:Header>
 <S:Body/>
</S:Envelope>
```

Appendix C.4 Retransmission

The RM Sourcediscovers that message number 2 was not accepted so it resends the message and requests an Acknowledgement:

```
<?xml version="1.0" encoding="UTF-8"?>
<S:Envelope xmlns:S="http://www.w3.org/2003/05/soap-envelope"
xmlns:wsrm="http://docs.oasis-open.org/ws-rx/wsrm/200702"
xmlns:wsa="http://www.w3.org/2005/08/addressing">
 <S:Header>
  <wsa:MessageID>
   http://Business456.com/guid/daa7d0b2-c8e0-476e-a9a4-d164154e38de
  </wsa:MessageID>
  <wsa:To>http://example.com/serviceB/123</wsa:To>
  <wsa:From>
   <wsa:Address>http://Business456.com/serviceA/789</wsa:Address>
  </wsa:From>
  <wsa:Action>http://example.com/serviceB/123/request</wsa:Action>
  <wsrm:Sequence>
```

```
   <wsrm:Identifier>http://Business456.com/RM/ABC</wsrm:Identifier>
   <wsrm:MessageNumber>2</wsrm:MessageNumber>
  </wsrm:Sequence>
  <wsrm:AckRequested>
   <wsrm:Identifier>http://Business456.com/RM/ABC</wsrm:Identifier>
  </wsrm:AckRequested>
 </S:Header>
 <S:Body>
  <!-- Some Application Data -->
 </S:Body>
</S:Envelope>
```

Appendix C.5 Termination

The RM Destination now responds with an Acknowledgement for the complete Sequence which can then be terminated:

```
<?xml version="1.0" encoding="UTF-8"?>
<S:Envelope xmlns:S="http://www.w3.org/2003/05/soap-envelope"
xmlns:wsrm="http://docs.oasis-open.org/ws-rx/wsrm/200702"
xmlns:wsa="http://www.w3.org/2005/08/addressing">
 <S:Header>
  <wsa:MessageID>
   http://example.com/guid/0baaf88d-483b-4ecf-a6d8-a7c2eb546811
  </wsa:MessageID>
  <wsa:To>http://Business456.com/serviceA/789</wsa:To>
  <wsa:From>
   <wsa:Address>http://example.com/serviceB/123</wsa:Address>
  </wsa:From>
  <wsa:Action>
    http://docs.oasis-open.org/ws-rx/wsrm/200702/SequenceAcknowledgement
  </wsa:Action>
  <wsrm:SequenceAcknowledgement>
   <wsrm:Identifier>http://Business456.com/RM/ABC</wsrm:Identifier>
   <wsrm:AcknowledgementRange Upper="3" Lower="1"/>
  </wsrm:SequenceAcknowledgement>
 </S:Header>
 <S:Body/>
</S:Envelope>
```

Terminate Sequence

```
<?xml version="1.0" encoding="UTF-8"?>
<S:Envelope xmlns:S="http://www.w3.org/2003/05/soap-envelope"
xmlns:wsrm="http://docs.oasis-open.org/ws-rx/wsrm/200702"
xmlns:wsa="http://www.w3.org/2005/08/addressing">
 <S:Header>
  <wsa:MessageID>
   http://Business456.com/guid/0baaf88d-483b-4ecf-a6d8-a7c2eb546812
  </wsa:MessageID>
  <wsa:To>http://example.com/serviceB/123</wsa:To>
  <wsa:Action>
    http://docs.oasis-open.org/ws-rx/wsrm/200702/TerminateSequence
  </wsa:Action>
  <wsa:From>
   <wsa:Address>http://Business456.com/serviceA/789</wsa:Address>
  </wsa:From>
 </S:Header>
 <S:Body>
  <wsrm:TerminateSequence>
   <wsrm:Identifier>http://Business456.com/RM/ABC</wsrm:Identifier>
   <wsrm:LastMsgNumber> 3 </wsrm:LastMsgNumber>
  </wsrm:TerminateSequence>
 </S:Body>
</S:Envelope>
```

Terminate Sequence Response

```
<?xml version="1.0" encoding="UTF-8"?>
<S:Envelope xmlns:S="http://www.w3.org/2003/05/soap-envelope"
xmlns:wsrm="http://docs.oasis-open.org/ws-rx/wsrm/200702"
xmlns:wsa="http://www.w3.org/2005/08/addressing">
 <S:Header>
  <wsa:MessageID>
   http://Business456.com/guid/0baaf88d-483b-4ecf-a6d8-a7c2eb546813
  </wsa:MessageID>
  <wsa:To>http://example.com/serviceA/789</wsa:To>
  <wsa:Action>
    http://docs.oasis-open.org/ws-rx/wsrm/200702/TerminateSequenceResponse
  </wsa:Action>
  <wsa:RelatesTo>
    http://Business456.com/guid/0baaf88d-483b-4ecf-a6d8-a7c2eb546812
  </wsa:RelatesTo>
  <wsa:From>
   <wsa:Address>http://Business456.com/serviceA/789</wsa:Address>
  </wsa:From>
 </S:Header>
 <S:Body>
  <wsrm:TerminateSequenceResponse>
   <wsrm:Identifier>http://Business456.com/RM/ABC</wsrm:Identifier>
  </wsrm:TerminateSequenceResponse>
 </S:Body>
</S:Envelope>
```

Appendix D. State Tables

This appendix specifies the non-normative state transition tables for RM Source and RM Destination.

The state tables describe the lifetime of a Sequence in both the RM Source and the RM Destination

Legend:

The first column of these tables contains the motivating event and has the following format:

Event
Event name [source] {ref}

Where:

- Event Name: indicates the name of the event. Event Names surrounded by "<>" are optional as described by the specification.
- [source]: indicates the source of the event; one of:
  - [msg] a Received message
  - [int]: an internal event such as the firing of a timer
  - [app]: the application
  - [unspec]: the source is unspecified

Each event / state combination cell in the tables in this appendix has the following format:

State Name
Action to take [next state] {ref}

Where:

- action to take: indicates that the state machine performs the following action. Actions surrounded by "<>" are optional as described by the specification. "Xmit" is used as a short form for the word "Transmit"
- [next state]: indicates the state to which the state machine will advance upon the performance of the action. For ease of reading the next state "same" indicates that the state does not change.
- {ref} is a reference to the document section describing the behavior in this cell

"N/A" in a cell indicates a state / event combination self-inconsistent with the state machine; should these conditions occur, it would indicate an implementation error. A blank cell indicates that the behavior is not described in this specification and does not indicate normal protocol operation. Implementations MAY generate a Sequence Terminated fault (see section 4.2) in these circumstances. Robust implementations MUST be able to operate in a stable manner despite the occurrence of unspecified event / state combinations.

Table 1 RM Source Sequence State Transition Table

Events	**Sequence States**					
	None	**Creating**	**Created**	**Closing**	**Closed**	**Terminating**
Create Sequence [unspec] {3.4}	Xmit Create Sequence [Creating] {3.4}	N/A	N/A	N/A	N/A	N/A
Create Sequence Response [msg] {3.4)		Process Create Sequence Response [Created] {3.4}				
Create Sequence Refused Fault [msg] {3.4}		No action [None] {4.6}				
Send message [app] {2.1}	N/A	N/A	Xmit message [Same] {2}	No action [Same] {2}	N/A	N/A
Retransmit of un-ack'd message [int]	N/A	N/A	Xmit message [Same] {2.3}	Xmit message [Same] {2.3}	N/A	N/A
SeqAck (non-final) [msg] {3.9}	Generate Unknown Sequence Fault [Same] {4.3}	Generate Unknown Sequence Fault [Same] {4.3}	Process Ack ranges [Same] {3.9}	Process Ack ranges [Same] {3.9}	Process Ack ranges [Same] {3.9}	Process Ack ranges [Same] {3.9}
Nack [msg] {3.9)	Generate Unknown Sequence Fault [Same] {4.3}	Generate Unknown Sequence Fault [Same] {4.3}	<Xmit message(s)> [Same] {3.9}	<Xmit message(s)> [Same] {3.9}	No action [Same]	No action [Same]
Message Number Rollover Fault [msg]	Generate Unknown Sequence Fault [Same] {4.3}	Generate Unknown Sequence Fault [Same] {4.3}	No action [Same]	No action [Same]	No action [Same]	No action [Same]
CloseSequence [msg] {3.5}	Generate Unknown Sequence Fault [Same] {4.3}	Generate Unknown Sequence Fault [Same] {4.3}	Xmit CloseSequence Response [Closed] {3.5}	Xmit CloseSequence Response [Closed] {3.5}	Xmit CloseSequence Response [Closed] {3.5}	Generate Unknown Sequence Fault [Same] {4.3}
<Close Sequence> [int] {3.5}	N/A		Xmit Close Sequence [Closing] {3.5}	N/A	N/A	N/A
Close Sequence Response [msg] {3.5}	Generate Unknown Sequence Fault [Same] {4.3}	Generate Unknown Sequence Fault [Same] {4.3}		No action [Closed] {3.5}	No action [Same] {3.5}	No action [Same] {3.5}

Events	**Sequence States**					
	None	**Creating**	**Created**	**Closing**	**Closed**	**Terminating**
SeqAck (final) [msg] {3.9}	Generate Unknown Sequence Fault [Same] {4.3}	Generate Unknown Sequence Fault [Same] {4.3}	Process Ack ranges [Closed] {3.9}	Process Ack ranges [Closed] {3.9}	Process Ack ranges [Same]	Process Ack ranges [Same]
Sequence Closed Fault [msg] {4.7}	Generate Unknown Sequence Fault [Same] {4.3}	Generate Unknown Sequence Fault [Same] {4.3}	No action [Closed] {4.7}	No action [Closed] {4.7}	No action [Same]	No action [Same]
Unknown Sequence Fault [msg] {4.3}			Terminate Sequence [None] {4.3}	Terminate Sequence [None] {4.3}	Terminate Sequence [None] {4.3}	Terminate Sequence [None] {4.3}
Sequence Terminated Fault [msg] {4.2}	N/A		Terminate Sequence [None] {4.2}	Terminate Sequence [None] {4.2}	Terminate Sequence [None] {4.2}	Terminate Sequence [None] {4.2}
TerminateSequence [msg] {3.6}	Generate Unknown Sequence Fault [Same] {4.3}	Generate Unknown Sequence Fault [Same] {4.3}	Xmit Terminate Sequence Response [None] {3.6}	Xmit Terminate Sequence Response [None] {3.6}	Xmit Terminate Sequence Response [None] {3.6}	Generate Unknown Sequence Fault [Same] {4.3}
Terminate Sequence [int]	N/A	No action [None] {unspec}	Xmit Terminate Sequence [Terminating]	Xmit Terminate Sequence [Terminating]	Xmit Terminate Sequence [Terminating]	N/A
Terminate Sequence Response [msg]	Generate Unknown Sequence Fault [Same] {4.3}	Generate Unknown Sequence Fault [Same] {4.3}				Terminate Sequence [None] {3.6}
Expires exceeded [int]	N/A	Terminate Sequence [None] {3.4}	Terminate Sequence [None] {3.4}	Terminate Sequence [None] {3.4}	Terminate Sequence [None] {3.4}	Terminate Sequence [None] {3.4}
Invalid Acknowledgement [msg] {4.4}	Generate Unknown Sequence Fault [Same] {4.3}	Generate Unknown Sequence Fault [Same] {4.3}	Generate Invalid Acknowledgeme nt Fault [Same] {4.4}	Generate Invalid Acknowledgeme nt Fault [Same] {4.4}	Generate Invalid Acknowledgeme nt Fault [Same] {4.4}	Generate Invalid Acknowledgeme nt Fault [Same] {4.4}

2134 Table 2 RM Destination Sequence State Transition Table

Events	**Sequence States**			
	None	**Created**	**Closed**	**Terminating**
CreateSequence (successful) [msg/int] {3.4}	Xmit Create Sequence Response [Created] {3.4}	N/A	N/A	

Events	**Sequence States**			
	None	**Created**	**Closed**	**Terminating**
CreateSequence (unsuccessful) [msg/int] {3.4}	Generate Create Sequence Refused Fault [None] {3.4}	N/A	N/A	
Message (with message number within range) [msg]	Generate Unknown Sequence Fault [Same] {4.3}	Accept Message; <Xmit SeqAck> [Same]	Generate Sequence Closed Fault (with SeqAck+Final) [Same] {3.5}	Generate Sequence Terminated Fault [Same] {4.2}
Message (with message number outside of range) [msg]	Generate Unknown Sequence Fault [Same] {4.3}	Xmit Message Number Rollover Fault [Same] {3.7}{4.5}	Generate Sequence Closed Fault (with SeqAck+Final) [Same] {3.5}	Generate Sequence Terminated Fault [Same] {4.2}
<AckRequested> [msg] {3.8}	Generate Unknown Seq Fault [Same] {4.3}	Xmit SeqAck [Same] {3.8}	Xmit SeqAck+Final [Same] {3.9}	Generate Sequence Terminated Fault [Same] {4.2}
CloseSequence [msg] {3.5}	Generate Unknown Sequence Fault [Same] {4.3}	Xmit CloseSequence Response with SeqAck+Final [Closed] {3.5}	Xmit CloseSequence Response with SeqAck+Final [Closed] {3.5}	Generate Sequence Terminated Fault [Same] {4.2}
<CloseSequence autonomously> [int]		Xmit CloseSequence with SeqAck+Final [Closed] {3.5}	Xmit CloseSequence with SeqAck+Final [Same] {3.5}	
CloseSequenceResponse [msg] {3.5}	Generate Unknown Sequence Fault [Same] {4.3}		No Action [Closed] {3.5}	Generate Sequence Terminated Fault [Same] {4.2}
TerminateSequence [msg] {3.6)	Generate Unknown Sequence Fault [Same] {4.3}	Xmit Terminate Sequence Response [None] {3.6}	Xmit Terminate Sequence Response [None] {3.6}	Xmit Terminate Sequence Response [None] {3.6}
<TerminateSequence autonomously> [int]		Xmit TerminateSequence with SeqAck+Final [Terminating] {3.6}	Xmit TerminateSequence with SeqAck+Final [Terminating] {3.6}	Xmit TerminateSequence with SeqAck+Final [Terminating] {3.6}
TerminateSequenceRespons e [msg]	Generate Unknown Sequence Fault [Same] {4.3}			Terminate Sequence [None]
UnknownSequence Fault [msg] {4.3}		Terminate Sequence [None] {4.3}	Terminate Sequence [None] {4.3}	Terminate Sequence [None] {4.3}
SequenceTerminated Fault [msg] {4.2}		Terminate Sequence [None] {4.2}	Terminate Sequence [None] {4.2}	Terminate Sequence [None] {4.3}
Invalid Acknowledgement Fault **[msg]** **{4.4}**	N/A			
Expires exceeded [int]	N/A	Terminate Sequence [None]	Terminate Sequence [None]	

Events	Sequence States			
	None	Created	Closed	Terminating
		{3.4}	{3.4}	
<Seq Acknowledgement autonomously> [int] {3.9}	N/A	Xmit SeqAck [Same] {3.9}	Xmit SeqAck+Final [Same] {3.9}	
Non WSRM message when WSRM required [msg] {4.8}	Generate WSRMRequired Fault [Same] {4.8}	Generate WSRMRequired Fault [Same] {4.8}	Generate WSRMRequired Fault [Same] {4.8}	

Appendix E. Acknowledgments

This document is based on initial contribution to OASIS WS-RX Technical Committee by the following authors:

- Ruslan Bilorusets, BEA
- Don Box, Microsoft
- Luis Felipe Cabrera, Microsoft
- Doug Davis, IBM
- Donald Ferguson, IBM
- Christopher Ferris, IBM
- Tom Freund. IBM
- Mary Ann Hondo, IBM
- John Ibbotson, IBM
- Lei Jin, BEA
- Chris Kaler, Microsoft
- David Langworthy-Editor, Microsoft
- Amelia Lewis, TIBCO Software
- Rodney Limprecht, Microsoft
- Steve Lucco, Microsoft
- Don Mullen, TIBCO Software
- Anthony Nadalin, IBM
- Mark Nottingham, BEA
- David Orchard, BEA
- Jamie Roots, IBM
- Shivajee Samdarshi, TIBCO Software
- John Shewchuk, Microsoft
- Tony Storey, IBM

The following individuals have provided invaluable input into the initial contribution:

- Keith Ballinger, Microsoft
- Stefan Batres, Microsoft
- Rebecca Bergersen, Iona
- Allen Brown, Microsoft
- Michael Conner, IBM
- George Copeland, Microsoft
- Francisco Curbera, IBM
- Paul Fremantle, IBM
- Steve Graham, IBM
- Pat Helland, Microsoft
- Rick Hill, Microsoft
- Scott Hinkelman, IBM
- Tim Holloway, IBM
- Efim Hudis, Microsoft
- David Ingham, Microsoft
- Gopal Kakivaya, Microsoft
- Johannes Klein, Microsoft
- Frank Leymann, IBM
- Martin Nally, IBM
- Peter Niblett, IBM
- Jeffrey Schlimmer, Microsoft
- James Snell, IBM
- Keith Stobie, Microsoft
- Satish Thatte, Microsoft
- Stephen Todd, IBM
- Sanjiva Weerawarana, IBM
- Roger Wolter, Microsoft

The following individuals were members of the committee during the development of this specification:

- Abbie Barbir, Nortel
- Charlton Barreto, Adobe
- Stefan Batres, Microsoft
- Hamid Ben Malek, Fujitsu
- Andreas Bjarlestam, Ericsson
- Toufic Boubez, Layer 7
- Doug Bunting, Sun
- Lloyd Burch, Novell
- Steve Carter, Novell
- Martin Chapman, Oracle
- Dave Chappell, Sonic
- Paul Cotton, Microsoft
- Glen Daniels, Sonic
- Doug Davis, IBM
- Blake Dournaee, Intel
- Jacques Durand, Fujitsu
- Colleen Evans, Microsoft
- Christopher Ferris, IBM
- Paul Fremantle, WSO2
- Robert Freund, Hitachi
- Peter Furniss, Erebor
- Marc Goodner, Microsoft
- Alastair Green, Choreology
- Mike Grogan, Sun
- Ondrej Hrebicek, Microsoft
- Kazunori Iwasa, Fujitsu
- Chamikara Jayalath, WSO2
- Lei Jin, BEA
- Ian Jones, BTplc
- Anish Karmarkar, Oracle
- Paul Knight, Nortel
- Dan Leshchiner, Tibco
- Mark Little, JBoss
- Lily Liu, webMethods
- Matt Lovett, IBM
- Ashok Malhotra, Oracle
- Jonathan Marsh, Microsoft
- Daniel Millwood, IBM

Jeff Mischkinsky, Oracle
Nilo Mitra, Ericsson
Peter Niblett, IBM
Duane Nickull, Adobe
Eisaku Nishiyama, Hitachi
Dave Orchard, BEA
Chouthri Palanisamy, NEC
Sanjay Patil, SAP
Gilbert Pilz, BEA
Martin Raepple, SAP
Eric Rajkovic, Oracle
Stefan Rossmanith, SAP
Tom Rutt, Fujitsu
Rich Salz, IBM
Shivajee Samdarshi, Tibco
Vladimir Videlov, SAP
Claus von Riegen, SAP
Pete Wenzel, Sun
Steve Winkler, SAP
Ümit Yalçinalp, SAP
Nobuyuki Yamamoto, Hitachi