1 Reliable Messaging

Reliable Messaging defines an interoperable protocol such that any two Messaging Service Handlers (MSH) can "reliably" exchange messages that are sent using "reliable messaging" delivery semantics.

"Reliably" means that the *From Party* can be highly certain that the message sent will be delivered to the *To Party*. If there is a problem in sending a message then the sender resends the message until either the message is delivered, or the sender gives up. If the message cannot be delivered, for example because there has been a catastrophic failure of the *To Party's* system, then the *From Party* is informed.

1.1 Persistent Storage and System Failure

A MSH that supports Reliable Messaging MUST keep messages, and/or selected data from these messages, in *persistent storage*. In this context *persistent storage* is a method of storing data that does not lose information after a system failure or interruption.

This specification recognizes that different degrees of resilience may be realized depending on the technology that is used to persist the data. However, as a minimum, persistent storage that has the resilience characteristics of a hard disk (or equivalent) SHOULD be used. It is strongly RECOMMENDED though that implementers of this specification use technology that is resilient to the failure of any single hardware or software component.

Even after a system interruption or failure, a MSH MUST ensure that messages in persistent storage are processed as if the system failure or interruption had not occurred. How this is done is an implementation decision.

In order to support the filtering of duplicate messages, a Receiving MSH SHOULD, save the ***MessageId*** in *persistent storage*. It is also RECOMMENDED that the following be kept in *Persistent Storage*:

- the complete message, at least until the information in the message has been passed to the application or other process that needs to process it
- the time the message was received, so that the information can be used to generate the response to a Message Status Request (see section **Error! Reference source not found.**)

1.2 Reliable Messaging Parameters

This section describes the parameters required to control reliable messaging. This parameter information is contained in the following:

- the *ebXML Message Header*, or
- the *CPA* that governs the processing of a message.

The table below indicates where these parameters may be set.

Parameter	CPA	Header
deliverySemantics	Yes	Yes
syncReplyMode	Yes	Yes
timeToLive	Yes	Yes
reliableMessagingMethod	No	Yes

Parameter	**CPA**	**Header**
intermediateAckRequested<DB> Should be just "ackRequested" </DB>	No	Yes
timeout	Yes	No
retries	Yes	No
retryInterval	Yes	No
reliableMessagingSupported	Yes	No
persistDuration	Yes	No

In this table, the following interpretation of the columns should be used:

1) if the **CPA** column contains a **Yes** then it indicates that the value that is present in the CPA determines the processing semantics

2) if the **CPA** column contains a **No** then it indicates that the parameter value is never specified in the **CPA**

3) if the **Header** column contains a **Yes** then it indicates that the parameter value MAY be specified in the *ebXML Header* document.

<DB> It is not clear what happens if a parameter is in both the CPA and the Header (parameters deliverySemantics, syncReplyMode, timeToLive). The above seems to suggest that if the value is in the header then it would be ignored.</DB>

These parameters are described below.

1.2.1 Delivery Semantics

The ***deliverySemantics*** parameter may be present as either <DB>in the CPA or as ??</DB>an attribute within the ***QualityOfService*** element of the ***ebXMLHeader*** document. The deliverySemantics attribute takes its value <DB>Does this mean that it has exactly the same value as the parameter in the CPA and it is copied into the header as a convenience to the MSH instead of the MSH having to look up value in the CPA. What happens, though, if the value in the CPA happens to be different from the value in the CPA. </DB>from the CPA that governs the processing of a given message. See section **Error! Reference source not found.** for more information.

1.2.2 Sync Reply Mode

The ***syncReplyMode*** parameter may be present as either an element within the ***ebXMLHeader*** element or as a parameter within the CPA. See section **Error! Reference source not found.** for more information.

1.2.3 Time To Live

The ***TimeToLive*** element may be presente within the ***ebXMLHeader*** document see section **Error! Reference source not found.** for more information.

1.2.4 Reliable Messaging Method

The ***ReliableMessagingMethod*** parameter indicates the requested method for Reliable Messaging that will be used when sending a Message. Valid values are:

- ***ebXML*** in this case the ebXML Reliable Messaging Protocol as defined in section 1.3.1 is followed, or

- ***Transport***, in this case a reliable transport protocol is used for reliable delivery of the message, see section 0<DB>This section has been removed therefore this is inconsistent.</DB>.

1.2.5 Intermediate Ack Requested

The ***IntermediateAckRequested*** parameter is used by the Sending MSH to request that the Receiving MSH that receives the *Message* returns an *acknowledgment message* with an ***Acknowledgment*** element with a ***type*** of ***IntemediateAcknowledgment***..

<DB>Do we define anywhere what is an acknowledgement message or do we rely on the Glossary?</DB>

Valid values for ***IntermediateAckRequested*** are:

- ***Unsigned*** - requests that an unsigned Delivery Receipt is requested
- ***Signed*** - requests that a signed Delivery Receipt is requested, or
- ***None*** - indicates that no Delivery Receipt is requested.

<DB>Replace Delivery Receipt by Intermediate Acknowledgement in the above. This imistake is also in the current version of the spec.</DB>

The default value is ***None***.

1.2.6 Timeout Parameter

The ***timeout*** parameter is an integer value that specifies the time in < seconds DB>Perhaps this should be an XML Schema TimeDuration. </DB>that the Sending MSH MUST wait for an *Acknowledgment Message* before first resending a message to the Receiving MSH.

1.2.7 Retries Parameter

The ***retries*** Parameter is an integer value that specifies the maximum number of times a Sending MSH SHOULD attempt to redeliver an unacknowledged or undelivered *message*.<DB>This should say per Communication Protocol.</DB>

1.2.8 RetryInterval Parameter

The ***retryInterval*** parameter is an integer value specifying, in seconds, DB>Perhaps this should be an XML Schema TimeDuration </DB>the time the Sending MSH SHOULD wait between retries, if an *Acknowledgment Message* is not received.<DB>The current version says MUST rather than SHOULD. A simple SHOULD suggests that it is OK to resend it earlier. Suggest saying that the time is minimum that the MSH MUST wait.</DB>

1.2.9 Reliable Messaging Methods Supported

The ***reliableMessagingMethodsSupported*** parameter is a list of the methods that a MSH uses to support Reliable Messaging. It must be a URI. The URI for the ebXML Reliable Messaging Protocol described in section 1.3.1 is ***http://www.ebxml.org/namespaces/reliableMessaging*** ***<DB>This is only every used in the CPA. Therefore it really does not need to be here.</DB>***

1.2.10 PersistDuration

The ***persistDuration*** parameter is specified in the CPA. <DB>We don't need to say this as it is stated in the table.</DB> It represents the minimum length of time, expressed as a [XMLSchema] timeDuration, that data from a *Message* that is sent reliably, is kept in *Persistent Storage* by a MSH that receives that *Message*. Note that implementations may determine that a message is persisted for longer than the time specified in ***persistDuration***, for example in order to meet legal requirements or the needs of a business process. This information is recorded separately within the CPA.

<DB>There seems to have been a lot of text cut out from the description of PersistDuration. There was a discussion on the list about how PersistDuration should described in the spec which

led to an agreed definition. We should reconsider including that text. Speciifically we should re-insert the followin ...

*"A MSH SHOULD NOT resend a message with the same **MessageId** to a receiving MSH if the elapsed time indicated by **persistDuration** has passed since the message was first sent as the receiving MSH will probably not treat it as a duplicate"*

</DB>

1.3 Methods of Implementing Reliable Messaging

Support for Reliable Messaging can be implemented in one of the following two ways:

- using the ebXML Reliable Messaging protocol, or
- using ebXML Header and Message structures together with commercial software products that are designed to provide reliable delivery of messages using alternative protocols

Use of alternative protocols to effect reliable delivery of messages is outside the scope of this specification.

<DB>If we provide absolutely no guidance on how to use alternative protocols then we run the risk of failing to get interoperability. For example, can we assume that the meaning of all the parameters (e.g. IntermediateAckRequested) is exactly the same whether we are using the ebXML reliable messaging protocol or not. Right?.</DB>

1.3.1 ebXML Reliable Messaging Protocol

The ebXML Reliable Messaging Protocol described in this section MUST be followed if the ***deliverySemantics*** parameter/element is set to ***OnceAndOnlyOnce*** and the ***ReliableMessagingMethod*** parameter/element is set to ***ebXML*** (the default).

The ebXML Reliable Messaging Protocol is illustrated by the figure below.

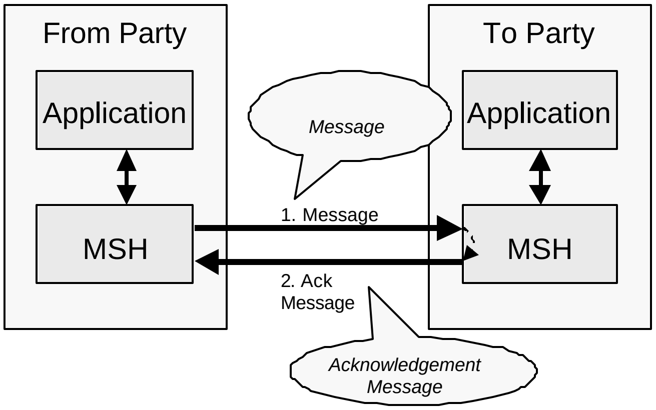

Figure 1-1 Indicating that a message has been received

The receipt of the *acknowledgment message* indicates that a *message* has been successfully received, and either processed or persisted by the receiving MSH to which the *message* was sent.

An *acknowledgment message* MUST contain a ***MessageData*** element with a ***RefToMessageId*** that contains the same value as the ***MessageId*** element in the *message being acknowledged*.

1.3.1.1 Sending Message Behavior

If a MSH is given data by an application that needs to be sent reliably then the MSH MUST do the following:

1) Create a message from components received from the application that includes:
    a) deliverySemantics set to OnceAndOnlyOnce, and
    b) a RoutingHeader element that identifies the sender and the receiver URIs
2) Save the message in *persistent storage* (see section 1.1)
3) Send the message to the Receiver MSH
4) Wait for the *Receiver* MSH to return an *acknowledgment message* and, if it does not, then resend the *identical* message as described in section 1.3.1.4

1.3.1.2 Receiving Message Behavior

If ***deliverySemantics*** on the received message is set to ***OnceAndOnlyOnce*** then do the following:

1) Check to see if the message is a duplicate (e.g. there is a message in *persistent storage* that was received earlier that contains the same value for the ***MessageId***)
2) If the message is not a duplicate then do the following:
    a) Save the ***MessageId*** of the received message in *persistent storage*. As an implementation decision, the whole message MAY be stored if there are other reasons for doing so
    b) If the received message contains a ***RefToMessageId*** element then do the following:
        i) Look for a message in *persistent storage* that has a ***MessageId*** that is the same as the value of ***RefToMessageId*** on the received Message
        ii) If a message is found in *persistent storage* then mark the persisted message as delivered
    c) <DB>What is entirely missing from here (and I can't find it anywhere else) is the requirement to send an acknowledgement message if the message isn't a duplicate !!! See updated text on Service and Action Element Values </DB>
3) If the message is a duplicate, then do the following:
    a) Look in persistent storage for a response to the received message (i.e. it contains a ***RefToMessageId*** that matches the ***MessageId*** of the received message)
    b) If no message was found in *persistent storage* then ignore the received message as either no message was generated in response to the message, or the processing of the earlier message is not yet complete
    c) If a message was found in *persistent storage* then resend the persisted message back to the MSH that sent the received message.

<DB>This assumes there is only one message that has been generated and persisted as a result of receiving an earlier message. There could be more. For example you could send an *acknowledgement message* followed later by a message that contained a business response. So you have to say either:

- the first message sent in reply,

- the most recent message, or
- leave it undefined.

I prefer the most recent as it will be more useful to get the business/process response than the acknowledgement.</DB>

1.3.1.3 Service and Action Element Values

<DB>Suggest renaming this to Generating an Acknowkledgement Message and including description of how to generate an acknowledgement with precise rules on what it contains.</DB>

An ***Acknowledgment*** element can be included in an ***ebXMLHeader*** that is part of a *message* that is being sent as a result of processing of an earlier message. In this case the values for the ***Service*** and ***Action*** elements are set by the designer of the Service (see section **Error! Reference source not found.**).

<DB>Later parts of this spec indicate that an Acknowledgement element can only be used with multi-hop. This is inconsistent. It is much simpler if the rule is if the Routing Header contains an ***ackRequested*** set to ***True*** then return an Acknowledgement element. This apparent restriction also complicates the use of syncReplyMode.</DB>

An ***Acknowledgment*** element also can be included in an ***ebXMLHeader*** that does not include any results from the processing of an earlier message. In this case, the values of the ***Service*** and ***Action*** elements MUST be set as follows:

- The ***Service*** element MUST be set to: ***http://www.ebxml.org/namespaces/messageService/MessageAcknowledgment***
- The ***Action*** element MUST be set to the value of the ***type*** attribute in the ***Acknowledgment*** element.<DB>This is now inconsistent as we no longer have delivery receipts as a valid type of acknowledgement.</DB>

1.3.1.4 Resending Lost Messages and Duplicate Filtering

This section describes the behavior that is required by the sender and receiver of a message in order to handle when messages are lost. A message is "lost" when a sending MSH does not receive a response to a message. For example, it is possible that a *message* was lost, for example:

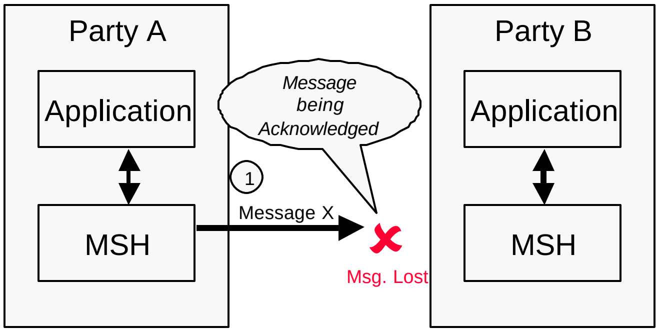

Figure 1-2 Lost Message

It is also possible that the *Acknowledgment Message* was lost, for example:

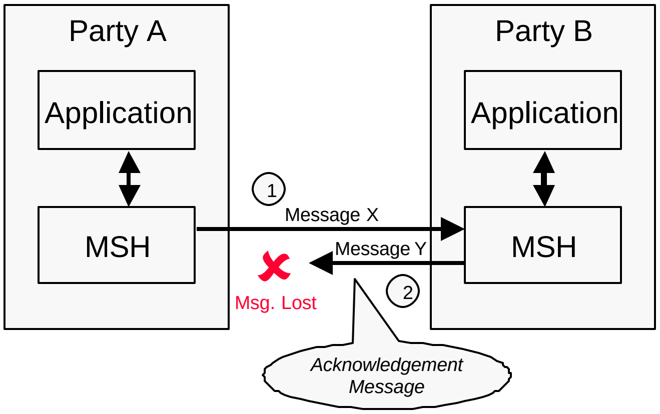

Figure 1-3 Lost Acknowledgment Message

The rules that apply are as follows:

1) The Sending MSH MUST resend the original message if an *Acknowledgment Message* has not been received from the Receiving MSH and either of the following are true:
    a) The message has not yet been resent and at least the time specified in the ***timeout*** parameter has passed since the first message was sent, or
    b) The message has been resent, and the following are both true:
        i) At least the time specified in the ***retryInterval*** has passed since the last time the message was resent, and
        ii) The message has been resent less than the number of times specified in the ***retries*** Parameter
2) If the Sending MSH does not receive an *Acknowledgment Message* after the maximum number of retries, the Sending MSH SHOULD notify the application and/or system administrator function.
3) If the Sending MSH detects a communications protocol error that is unrecoverable at the transport protocol level then the Sending MSH SHOULD first attempt to resend the message using the same transport protocol until the number of ***retries*** has been reached, and then again, using a different communications protocol<DB>We should allow multiple different communication protocols and not just one. This is also in the current version of the spec</DB>, if the CPA allows this. If these are not successful, then notify the From Party of the failure to deliver as described in section 1.4.

1.3.2 Duplicate Message Handling

In this context:

- an *identical message* is a *message* that contains the exact same *ebXML Header* and *ebXML Payload* as the earlier *message* that was sent previously.
- a *duplicate message* is a *message* that contains the same ***MessageId*** as an earlier message that was received.
- <DB>In the last version of the spec there was a noted disagreement between Chris and myself around sending the most recent message. This has not been discussed and needs to be.</DB>

Note that the Communication Protocol Envelope MAY be different. This means that the same message MAY be sent using different communication protocols and the reliable messaging behavior described in this section will still apply. The ability to use alternative communication protocols is specified in the CPA and is an OPTIONAL implementation specific feature.

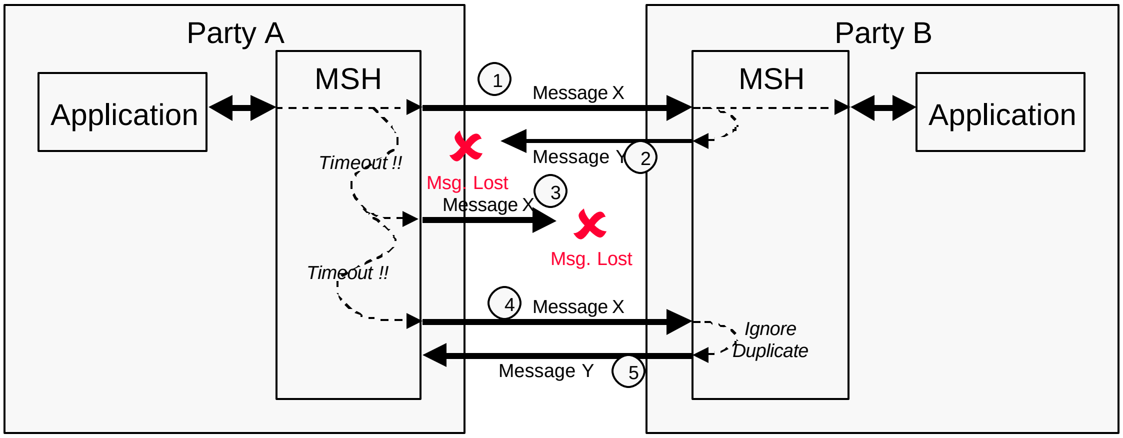

Figure 1-4 Resending Unacknowledged Messages

The diagram above shows the behavior that MUST be followed by the sending and receiving MSH for messages that require reliable delivery as regards to duplicate message receipt<DB>I think the phrase " that require reliable delivery as regards to duplicate message receipt" is vague. Suggest change to "that are sent with ***deliverySemantics*** of ***OnceAndOnlyOnce***. </DB>. Specifically:

1) The sender of the *message* (e.g. Party A) MUST re-send the *identical message* if no *Acknowledgment Message* is received

2) The recipient of the *message* (e.g. Party B), when it receives a *duplicate message*, MUST re-send to the sender of the *message* (e.g. Party A), a message identical to the *message* that was originally sent in response to the duplicate message

3) The recipient of a duplicate *message* MUST NOT forward them a second time to the application or other process that would normally be expected to process received messages.

1.3.2.1 Multi-hop Reliable Messaging

Multi-hop Reliable Messaging with Intermediate Acknowledgments is similar to Multi-hop Reliable Messaging without Intermediate Acknowledgment except that any of the Parties that are transmitting a Message can request that the recipient return an *Intermediate Acknowledgment.*

<DB>The above paragraph doesn't make sense now as:

1) Multi-hop messaging without intermediate acks has been removed

2) Delivery Receipt has been removed so that intermediate acks is now only acks.</DB>

This is illustrated by the diagram below.

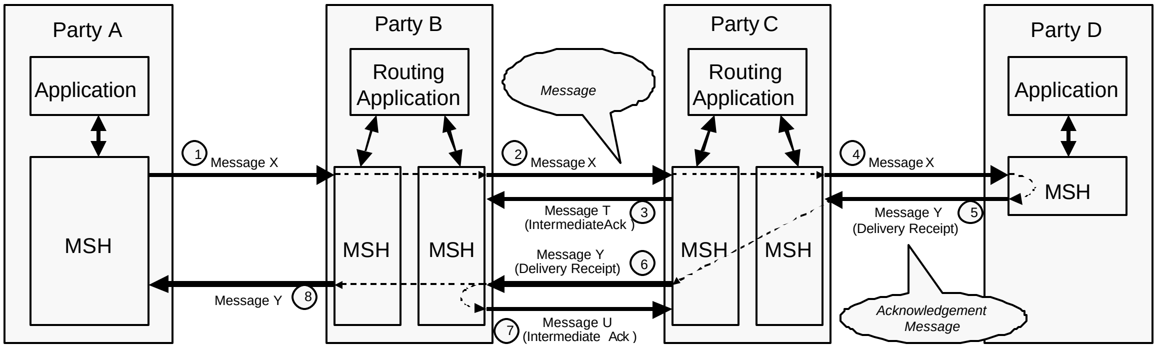

Figure 1-6 Multi-hop Reliable Messaging

<CBF>The image above needs to be fixed so that delivery receipt is not included. Intermediate acks only</CBF>

The rules that apply to Multi-hop Reliable Messaging are as follows:

- Any Party that is sending a message can request that the recipient send an *Acknowledgment Message* by setting the ***AckRequested*** of the ***RoutingHeader*** for the hop to ***Signed*** or ***Unsigned***.
- a MSH that is not the *To Party* receives a message that requires an Intermediate Acknowledgment then: the MSH MUST return an *Acknowledgment Message* with:
    i) The ***Service*** and ***Action*** elements set as in defined in section 1.1
    ii) The ***From*** element contains the ***ReceiverURI*** from the last ***RoutingHeader*** in the message that has just been received
    iii) The ***To*** element contains the ***SenderURI*** from the last ***RoutingHeader*** in the message that has just been received
    iv) a ***RefToMessageId*** element that contains the ***MessageId*** of the message being acknowledged
    v) a ***QualityOfServiceInfo*** element with ***deliverySemantics*** set to ***BestEffort***

<DB>This is now vague as the sender of a message may not know in advance whether they are sending a message to an intermediary</DB>

1.4 Failed Message Delivery

In the event that some actor<DB>Actor is not used as a term anywhere else in the spec. Do we really want to introduce it? </DB> is involved, in some capacity, in the delivery of a *message* has determined that a *message* cannot be delivered to the application or other process that has been designated to process the message, that actor SHOULD send a delivery failure notification *message* to the *From Party* that sent the *message*. The delivery failure notification message contains:

- a ***From Party*** that identifies the Party that detected the problem
- a ***To Party*** that identifies the ***From Party*** that created the message that could not be delivered
- a ***Service*** element and ***Action*** element set as described in **Error! Reference source not found.**
- a ***QualityOfServiceInfo*** element with ***deliverySemantics*** set to the same value as the ***deliverySemantics*** on the message that could not be delivered
- an ***Error*** element with a severity of:

- ***Error*** if the Party that detected the problem could not even transmit the message (e.g. Transmission 3 was impossible)<DB>There is now no diagram, so we need to change this.</DB>
- ***Warning*** if the message (e.g. Message X in Transmission 3) was transmitted, but no acknowledgment was received. This means that the message probably was not delivered although there is a small probability that it was
- an ***ErrorCode*** of ***DeliveryFailure***