



---

Creating A Single Global Electronic Market

## Message Service Specification

### ebXML Transport, Routing & Packaging

Version ~~0.91~~0.92

2 January 2001

# 1 Status of this Document

This document specifies an ebXML DRAFT for the eBusiness community Distribution of this document is unlimited.

The document formatting is based on the Internet Society's Standard RFC format converted to Microsoft Word 2000 format.

*This version*

[http://www.ebxml.org/working/project\\_teams](http://www.ebxml.org/working/project_teams)

*Latest version*

<http://www.ebxml.org>

*Previous version*

[http://www.ebxml.org/...](http://www.ebxml.org/)

## 2 ebXML Participants

The authors wish to acknowledge the support of the members of the Transport, Routing and Packaging Project Team who contributed ideas to this specification by the group's discussion email list, on conference calls and during face-to-face meeting.

Ralph Berwanger – bTrade.com  
Jonathan Borden – Author of XMTP  
Jon Bosak – Sun Microsystems  
Marc Breissinger – webMethods  
Dick Brooks – Group 8760  
Doug Bunting – Ariba  
David Burdett – Commerce One  
Len Callaway – Drummond Group, Inc.  
David Craft – VerticalNet  
Philippe De Smedt – Viquity  
Lawrence Ding – WorldSpan  
Rik Drummond – Drummond Group, Inc. (Representing XML Solutions)  
Christopher Ferris – Sun Microsystems  
Maryann Hondo – IBM  
Jim Hughes – Fujitsu  
John Ibbotson – IBM  
Ian Jones – British Telecommunications  
Ravi Kacker – Kraft Foods  
Nick Kassem – Sun Microsystems  
Henry Lowe – OMG  
Jim McCarthy – webXI  
Bob Miller – GSX  
Andrew Eisenberg – Progress Software  
Dale Moberg – Sterling Commerce  
Joel Munter – Intel  
Farrukh Najmi – Sun Microsystems  
Akira Ochi – Fujitsu  
Martin Sachs, IBM  
Masayoshi Shimamura – Fujitsu  
Kathy Spector – Extricity  
Nikola Stojanovic – Columbine JDS Systems  
Gordon Van Huizen – Process Software  
Martha Warfelt – Daimler Chrysler  
Prasad Yendluri – Web Methods

### 3 Table of Contents

1	Status of this Document .....	2
2	ebXML Participants .....	<del>33</del>
3	Table of Contents .....	<del>44</del>
4	Introduction .....	<del>99</del>
4.1	Summary of Contents of Document .....	<del>99</del>
4.2	Document Conventions .....	<del>1040</del>
4.3	Audience .....	<del>1040</del>
4.4	Caveats and Assumptions .....	<del>1040</del>
4.5	Related Documents .....	<del>1040</del>
5	Design Objectives .....	<del>1144</del>
6	System Overview .....	<del>1313</del> <del>1312</del>
6.1	What the Message Service does .....	<del>1313</del> <del>1312</del>
6.2	Message Service Overview .....	<del>1313</del> <del>1312</del>
7	Packaging Specification .....	<del>1515</del> <del>1514</del>
7.1	Introduction .....	<del>1515</del> <del>1514</del>
7.1.1	ebXML Header Envelope and ebXML Payload Envelope .....	<del>1515</del> <del>1514</del>
7.1.2	MIME usage Conventions .....	<del>1646</del> <del>1645</del>
7.2	ebXML Message Envelope .....	<del>1646</del> <del>1645</del>
7.2.1	Content-Type .....	<del>1646</del> <del>1645</del>
7.2.1.1	type Attribute .....	<del>1646</del> <del>1645</del>
7.2.1.2	boundary Attribute .....	<del>1646</del> <del>1645</del>
7.2.1.3	version Attribute .....	<del>1646</del> <del>1645</del>
7.2.2	ebXML Message Envelope Example .....	<del>1746</del> <del>1645</del>
7.3	ebXML Header Container .....	<del>1747</del> <del>1746</del>
7.3.1	Content-ID .....	<del>1747</del> <del>1746</del>
7.3.2	Content-Type .....	<del>1747</del> <del>1746</del>
7.3.2.1	version Attribute .....	<del>1747</del> <del>1746</del>
7.3.2.2	charset Attribute .....	<del>1747</del> <del>1746</del>
7.3.3	ebXML Header Envelope Example .....	<del>1848</del> <del>1846</del>
7.4	ebXML Payload Container .....	<del>1848</del> <del>1847</del>
7.4.1	Content-ID .....	<del>1948</del> <del>1847</del>
7.4.2	Content-Type .....	<del>1948</del> <del>1847</del>
7.4.3	Example of an ebXML MIME Payload Container .....	<del>1949</del> <del>1948</del>
7.5	Additional MIME Parameters .....	<del>1949</del> <del>1948</del>
7.6	Reporting MIME Errors .....	<del>1949</del> <del>1948</del>
8	ebXML Header Document .....	<del>2020</del> <del>2019</del>
8.1	XML Prolog .....	<del>2020</del> <del>2019</del>
8.1.1	XML Declaration .....	<del>2020</del> <del>2019</del>
8.1.2	Encoding Declaration .....	<del>2020</del> <del>2019</del>
8.1.3	Standalone Document Declaration .....	<del>2020</del> <del>2019</del>
8.1.4	Document Type Declaration .....	<del>2124</del> <del>2120</del>
8.2	ebXMLHeader Element .....	<del>2124</del> <del>2120</del>
8.2.1	ebXMLHeader attributes .....	<del>2124</del> <del>2120</del>
8.2.1.1	Namespace attribute .....	<del>2124</del> <del>2120</del>
8.2.1.2	version attribute .....	<del>2124</del> <del>2120</del>

8.2.2	ebXMLHeader elements .....	<a href="#">21242120</a>
8.2.3	Combining Principal Header Elements .....	<a href="#">22222220</a>
8.2.3.1	Manifest element .....	<a href="#">22222221</a>
8.2.3.2	Header element .....	<a href="#">22222221</a>
8.2.3.3	RoutingHeaderList element .....	<a href="#">22222221</a>
8.2.3.4	ApplicationHeaders element .....	<a href="#">22222221</a>
8.2.3.5	StatusData element .....	<a href="#">22222221</a>
8.2.3.6	ErrorList element .....	<a href="#">22222221</a>
8.2.3.7	Acknowledgment element .....	<a href="#">23232221</a>
8.2.3.8	Signature element .....	<a href="#">23232221</a>
8.2.3.9	#wildcard element content .....	<a href="#">23232221</a>
8.2.4	ebXMLHeader sample .....	<a href="#">23232322</a>
8.3	Manifest element .....	<a href="#">23232322</a>
8.3.1	Reference element .....	<a href="#">23232322</a>
8.3.1.1	Description element .....	<a href="#">24242422</a>
8.3.1.2	Schema element .....	<a href="#">24242422</a>
8.3.2	Manifest sample .....	<a href="#">25252423</a>
8.4	Header element .....	<a href="#">25252523</a>
8.4.1	From and To elements .....	<a href="#">25252523</a>
8.4.2	CPAId element .....	<a href="#">26262524</a>
8.4.3	ConversationId element .....	<a href="#">26262624</a>
8.4.4	Service element .....	<a href="#">27272624</a>
8.4.4.1	type attribute .....	<a href="#">27272624</a>
8.4.4.2	ebXML Message Service Header namespace .....	<a href="#">27272624</a>
8.4.5	Action element .....	<a href="#">27272624</a>
8.4.6	MessageData element .....	<a href="#">27272624</a>
8.4.6.1	MessageId element .....	<a href="#">27272725</a>
8.4.6.2	Timestamp element .....	<a href="#">27272725</a>
8.4.6.3	RefToMessageId element .....	<a href="#">28282725</a>
8.4.7	<del>ReliableMessagingInfoQualityOfServiceInfo</del> element .....	<a href="#">28282725</a>
8.4.7.1	deliverySemantics attribute .....	<a href="#">28282725</a>
8.4.7.2	DeliveryReceiptRequested attribute .....	<a href="#">29292826</a>
8.4.7.3	syncReplyMode attribute .....	<a href="#">30292826</a>
8.4.7.4	TimeToLive attribute .....	<a href="#">30292927</a>
8.4.8	Description element .....	<a href="#">31302927</a>
8.4.9	#wildcard element .....	<a href="#">31302927</a>
8.4.10	Header sample .....	<a href="#">31302927</a>
8.5	RoutingHeaderList element .....	<a href="#">32303028</a>
8.5.1	Routing Header Element .....	<a href="#">32313028</a>
8.5.1.1	SenderURI element .....	<a href="#">32313028</a>
8.5.1.2	ReceiverURI element .....	<a href="#">33313028</a>
8.5.1.3	ErrorURI element .....	<a href="#">33313028</a>
8.5.1.4	Timestamp element .....	<a href="#">33313128</a>
8.5.1.5	SequenceNumber element .....	<a href="#">33313129</a>
8.5.1.6	#wildcard .....	<a href="#">33323129</a>
8.5.2	Single Hop Routing Header Sample .....	<a href="#">34323229</a>
8.5.3	Multi-hop Routing Header Sample .....	<a href="#">35333330</a>
8.6	ApplicationHeaders Element .....	<a href="#">36343431</a>
8.6.1	ApplicationHeaders sample .....	<a href="#">36353431</a>
8.7	StatusData Element .....	<a href="#">36353431</a>
8.8	ErrorList Element .....	<a href="#">37353532</a>
8.8.1	id attribute .....	<a href="#">37353532</a>
8.8.2	highestSeverity attribute .....	<a href="#">37353532</a>
8.8.3	Error element .....	<a href="#">37363532</a>
8.8.3.1	codeContext attribute .....	<a href="#">37363532</a>
8.8.3.2	errorCode attribute .....	<a href="#">37363533</a>
8.8.3.3	severity attribute .....	<a href="#">37363533</a>
8.8.3.4	location attribute .....	<a href="#">38363633</a>
8.8.3.5	errorMessage attribute .....	<a href="#">38363633</a>
8.8.3.6	softwareDetails attribute .....	<a href="#">38373633</a>
8.8.4	Examples .....	<a href="#">38373634</a>

- 8.8.5 errorCode values ..... [39373734](#)
- 8.8.6 Reporting Errors in the ebXML Header Document ..... [39373734](#)
- 8.8.7 Non-XML Document Errors ..... [39383735](#)
- 8.9 Acknowledgment Element ..... [40393835](#)
  - 8.9.1 Timestamp element ..... [41393936](#)
  - 8.9.2 From element ..... [41393936](#)
  - 8.9.3 type attribute ..... [41393936](#)
  - 8.9.4 signed attribute ..... [41403936](#)
- 8.10 Signature Element ..... [41403936](#)
- 9 Message Service Handler Services ..... [42414037](#)
  - 9.1 Message Status Request Service ..... [42414037](#)
    - 9.1.1 Message Status Request Message ..... [42414037](#)
    - 9.1.2 Message Status Response Message ..... [42414037](#)
    - 9.1.3 Security Considerations ..... [43424138](#)
  - 9.2 Message Service Handler Ping Service ..... [43424138](#)
    - 9.2.1 Message Service Handler Ping Message ..... [43424138](#)
    - 9.2.2 Message Service Handler Pong Message ..... [43424138](#)
    - 9.2.3 Security Considerations ..... [44434239](#)
- 10 Reliable Messaging ..... [45444340](#)
  - 10.1.1 Persistent Storage and System Failure ..... [45444340](#)
  - 10.1.2 Methods of Implementing Reliable Messaging ..... [45444340](#)
  - 10.2 ebXML Reliable Messaging Protocol ..... [45444340](#)
    - 10.2.1 Single-hop Reliable Messaging ..... [46454441](#)
      - 10.2.1.1 Sending Message Behavior ..... [46454441](#)
      - 10.2.1.2 Receiving Message Behavior ..... [47464542](#)
      - 10.2.1.3 Resending Lost Messages ..... [48474643](#)
    - 10.2.2 Multi-hop Reliable Messaging ..... [50494845](#)
      - 10.2.2.1 Multi-hop Reliable Messaging without Intermediate Acknowledgments ..... [Error! Bookmark not defined.494845](#)
      - 10.2.2.2 Multi-hop Reliable Messaging with Intermediate Acknowledgments ..... [Error! Bookmark not defined.504946](#)
  - 10.3 ebXML Reliable Messaging using Queuing Transports ..... [50515047](#)
  - 10.4 Service and Action Element Values ..... [53525148](#)
  - 10.5 Failed Message Delivery ..... [54525148](#)
  - 10.6 Reliable Messaging Parameters ..... [55545249](#)
    - 10.6.1 Who sets Message Service Parameters ..... [55545249](#)
    - 10.6.2 From Party Parameters ..... [56555350](#)
      - 10.6.2.1 Delivery Semantics ..... [56555450](#)
      - 10.6.2.2 Delivery Receipt Requested ..... [56555451](#)
      - 10.6.2.3 Sync Reply Mode ..... [56555451](#)
      - 10.6.2.4 Time To Live ..... [57555451](#)
    - 10.6.3 To Party Parameters ..... [57555451](#)
      - 10.6.3.1 Delivery Receipt Provided ..... [57555451](#)
    - 10.6.4 Sending MSH Parameters ..... [57565451](#)
      - 10.6.4.1 Reliable Messaging Method ..... [57565451](#)
      - 10.6.4.2 Intermediate Ack Requested ..... [57565551](#)
      - 10.6.4.3 Timeout Parameter ..... [57565552](#)
      - 10.6.4.4 Retries Parameter ..... [58565552](#)
      - 10.6.4.5 RetryInterval Parameter ..... [58565552](#)
      - 10.6.4.6 Deciding when to resend a message ..... [58565552](#)
    - 10.6.5 Receiving MSH Parameters ..... [58575652](#)
      - 10.6.5.1 Reliable Messaging Methods Supported ..... [58575652](#)
      - 10.6.5.2 PersistDuration ..... [58575652](#)
      - 10.6.5.3 MSH Time Accuracy ..... [59585653](#)
- 11 Error Reporting and Handling ..... [60595754](#)
  - 11.1 Definitions ..... [60595754](#)

- 11.2 Types of Errors ..... [60595754](#)
- 11.3 When to generate Error Messages ..... [60595754](#)
  - 11.3.1 Security Considerations..... [60595754](#)
- 11.4 Identifying the Error Reporting Location..... [60595754](#)
- 11.5 Service and Action Element Values ..... [61605855](#)
- 12 Security ..... [62615956](#)
  - 12.1 Security and Management..... [67666456](#)
  - 12.2 Collaboration Party Profiles ..... [68676456](#)
  - 12.3 Risks ..... [68676456](#)
    - 12.3.1 Unauthorized Access..... [68676456](#)
    - 12.3.2 Data Integrity and Confidentiality..... [68676457](#)
    - 12.3.3 Denial-of Service ..... [68676457](#)
  - 12.4 Countermeasure Technologies ..... [69676457](#)
    - 12.4.1 ebXML Message Countermeasures for Unauthorized Access and Data Integrity .... [69676457](#)
    - 12.4.2 Digital Certificates..... [69676457](#)
    - 12.4.3 ebXML Message Countermeasures for Denial of Service..... [69686457](#)
    - 12.4.4 ebXML Management Countermeasures for Denial of Service..... [69686458](#)
  - 12.5 Profiles..... [69686458](#)
    - 12.5.1 XML Digital Signature (XMLDSIG) ..... [69686458](#)
    - 12.5.2 Profile - XML Signature signing of header and/or payload ..... [70686458](#)
      - 12.5.2.1 Risks ..... [70686458](#)
      - 12.5.2.2 Benefits ..... [70696458](#)
    - 12.5.3 S/MIME..... [70696458](#)
    - 12.5.4 Profile - S/MIME signing of message payload ..... [70696459](#)
      - 12.5.4.1 Sample S/MIME signed payload..... [71696459](#)
      - 12.5.4.2 Risks ..... [71706460](#)
      - 12.5.4.3 Benefits ..... [71706460](#)
    - 12.5.5 Profile - S/MIME encryption of message payload ..... [71706460](#)
      - 12.5.5.1 Risks ..... [72706460](#)
      - 12.5.5.2 Benefits ..... [72716460](#)
    - 12.5.6 PGP/MIME..... [72716460](#)
    - 12.5.7 Profile - PGP/MIME signing of message payload ..... [72716460](#)
      - 12.5.7.1 Risks ..... [72716460](#)
      - 12.5.7.2 Benefits ..... [72716460](#)
    - 12.5.8 Profile - PGP/MIME encryption of message payload..... [72716460](#)
      - 12.5.8.1 Risks ..... [72716461](#)
      - 12.5.8.2 Benefits ..... [72716461](#)
- 13 Synchronous and Asynchronous Responses..... [73726562](#)
- 14 References..... [74736663](#)
  - 14.1 Normative References..... [74736663](#)
  - 14.2 Non-Normative References ..... [74736663](#)
- 15 Disclaimer ..... [76756865](#)
- 16 Contact Information..... [77766966](#)
- Appendix A ebXMLHeader Schema and Data Type Definitions ..... [79787168](#)
  - A.1 Schema Definition ..... [79787168](#)
  - A.2 Data Type Definition ..... [87827572](#)
- Appendix B Examples..... [88837673](#)
- Appendix C Communication Protocol Interfaces..... [89847774](#)
  - C.1 HTTP ..... [89847774](#)

C.1.1	Asynchronous HTTP .....	<a href="#">89847774</a>
C.1.2	Synchronous HTTP .....	<a href="#">90857875</a>
C.2	SMTP.....	<a href="#">91867976</a>
C.3	FTP .....	<a href="#">92878077</a>
C.4	Communication Protocol Errors.....	<a href="#">92878077</a>
C.4.1	Use of Error Codes.....	<a href="#">92878077</a>
C.4.2	Communication Errors during Reliable Messaging .....	<a href="#">92878077</a>
Appendix D Reliable Messaging Processing Logic.....		<a href="#">93888178</a>
Copyright Statement .....		<a href="#">94898279</a>

## 1 4 Introduction

2 This is a draft standard for trial implementation. The specification is the one of a series of  
3 specifications. The main specification that is yet to be developed is the ebXML Service Interface  
4 specification that describes, in a language independent way, how an application or other process  
5 can interact with software that complies with this ebXML Message Service specification. The  
6 ebXML Service Interface specification is being developed as separate document. It will be  
7 included in a later version of this specification or as an additional specification.

### 8 4.1 Summary of Contents of Document

9 This specification defines the ebXML Message Service protocol that enables the secure and  
10 reliable exchange of messages between two parties. It includes descriptions of:

- 11 • the ebXML Message structure used to package payload data for transport between  
12 parties
- 13 • the behavior of the Message Service Handler that sends and receives those messages  
14 over a data communication protocol.

15 This specification is independent of both the payload and the communication protocol used,  
16 although Appendices to this specification describe how to use this specification with [HTTP] and  
17 [SMTP].

18 This specification is organized around the following topics:

- 19 • Packaging Specification – A description of how to package an ebXML Message and its  
20 associated parts into a form that can be placed into the body of a communications  
21 protocol such as HTTP or SMTP (section 7)
- 22 • Message Headers – A specification of the structure and composition of the information  
23 necessary for an ebXML Message Service to successfully generate or process an ebXML  
24 compliant message. This is represented as an XML document called the ebXML Header  
25 document (section 8)
- 26 • Message Service Handler Services – A description of two services that enable one  
27 service to discover the status of another Message Service Handler or an individual  
28 message (section 9)
- 29 • Reliable Messaging – The Reliable Messaging function defines an interoperable protocol  
30 such that any two Message Service implementations can “reliably” exchange messages  
31 that are sent using “reliable messaging” semantics (section 10)
- 32 • Error Handling – This section describes how one ebXML Message Service reports errors  
33 it detects to another ebXML Message Service Handler (section [11](#))
- 34 • Security – This provides a complete specification of the security requirements for ebXML  
35 Messages (~~section 0~~[section 12](#)).

36 Appendices to this specification cover the following:

- 37 • Appendix A Schemas and DTD Definitions – This contains [XML Schema] and [XML]  
38 Data Type Definitions for the ebXML Header document
- 39 • Appendix B Examples – This contains sample message content
- 40 • Appendix C Communication Protocol Envelope Mappings – This describes how to  
41 transport ebXML Message Service compliant messages over [HTTP] and [SMTP]
- 42 • Appendix D Reliable Messaging Protocol Logic – this non-normative appendix provides  
43 processing logic that describes the behavior of a Message Service Handler when sending  
44 or receiving messages with reliable delivery

## 4.2 Document Conventions

Terms in *Italics* are defined in the ebXML Glossary of Terms [Glossary]. Terms listed in **Bold Italics** represent the element and/or attribute content of the XML ebXMLHeader. Terms listed in Courier font relate to MIME components.

The keywords **MUST**, **MUST NOT**, **REQUIRED**, **SHALL**, **SHALL NOT**, **SHOULD**, **SHOULD NOT**, **RECOMMENDED**, **MAY**, and **OPTIONAL**, when they appear in this document, are to be interpreted as described in RFC 2119 [Bra97].

Note that the force of these words is modified by the requirement level of the document in which they are used.

- **MUST**: This word, or the terms "REQUIRED" or "SHALL", means that the definition is an absolute requirement of the specification.
- **MUST NOT**: This phrase, or the phrase "SHALL NOT", means that the definition is an absolute prohibition of the specification.
- **SHOULD**: This word, or the adjective "RECOMMENDED", means that there may exist valid reasons in particular circumstances to ignore a particular item, but the full implications must be understood and carefully weighed before choosing a different course.
- **SHOULD NOT**: This phrase, or the phrase "NOT RECOMMENDED", means that there may exist valid reasons in particular circumstances when the particular behavior is acceptable or even useful, but the full implications should be understood and the case carefully weighed before implementing any behavior described with this label.
- MAY: This word, or the adjective "OPTIONAL", mean that an item is truly optional. One vendor may choose to include the item because a particular marketplace requires it or because the vendor feels that it enhances the product while another vendor may omit the same item. An implementation which does not include a particular option MUST be prepared to interoperate with another implementation which does include the option, though perhaps with reduced functionality. In the same vein an implementation which does include a particular option MUST be prepared to interoperate with another implementation which does not include the option (except, of course, for the feature the option provides.)

## 4.3 Audience

The target audience for this specification is the community of software developers who will implement the ebXML Message Service.

## 4.4 Caveats and Assumptions

It is assumed that the reader has an understanding of transport protocols, MIME, ~~and~~ XML ~~and~~ security technologies.

## 4.5 Related Documents

The following set of related specifications will be delivered in phases:

- ebXML Message Services Requirements Specification [ebXMLMSREQ] – defines the requirements of the Message Services
- ebXML Technical Architecture Security Specification – defines the security mechanisms necessary to negate anticipated, selected threats
- **ebXML Collaboration Protocol Profile and Agreement Specification [ebXMLTP]** (under development) - defines how one party can discover and/or agree upon the information that party needs to know about another party prior to sending them a message that complies with this specification

- 91 • **ebXML Message Service Interface Specification** (to be developed) - defines an
- 92 interface that may be used by software to interact with an ebXML Message Service
- 93 ~~—ebXML Message Services Security Specification (under development)— defines the~~
- 94 ~~security mechanisms necessary to negate anticipated, selected threats~~
- 95 ~~—ebXML Message Services Requirements Specification [ebXMLMSREQ]— defines the~~
- 96 ~~requirements of the Message Services~~

## 97 **5 Design Objectives**

98 The design objectives of this specification are to define a Message Service (MS) to support XML  
99 based electronic business between small, medium and large enterprises. This specification is  
100 intended to enable a low cost solution, while preserving a vendor's ability to add unique value  
101 through added robustness and superior performance. It is the intention of the Transport, Routing  
102 and Packaging Project Team to keep this specification as straightforward and succinct as  
103 possible.

104 Every item in this specification will be prototyped by the ebXML Proof of Concept Team in order  
105 to ensure the clarity and accuracy of this specification.

106 **6 System Overview**

107 This document defines the ebXML Message Service (MS) component of the ebXML  
 108 infrastructure. The ebXML Message Service defines the message enveloping and header  
 109 document schema used to transfer ebXML Messages over a communication protocol such as  
 110 HTTP, SMTP, etc. This document provides sufficient detail to develop software for the packaging,  
 111 exchange and processing of ebXML Messages.

112 **6.1 What the Message Service does**

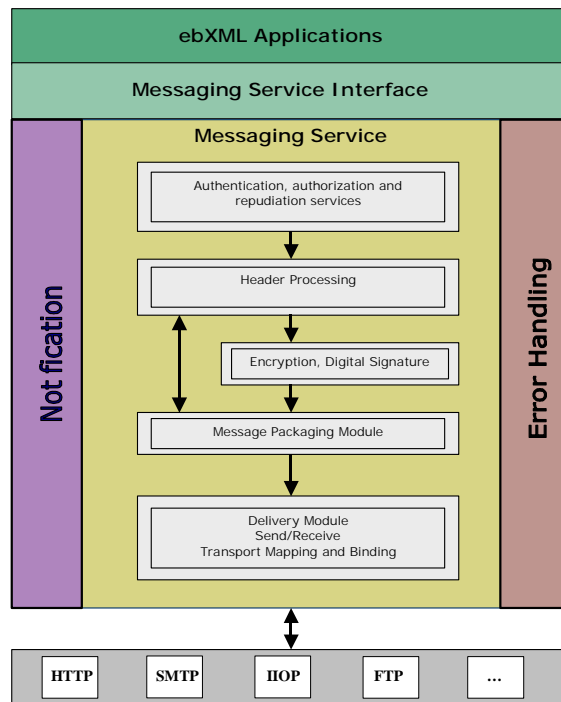
113 The ebXML Message Service defines robust, yet basic, functionality to transfer messages using  
 114 various existing communication protocols. The ebXML Message Service will perform in a manner  
 115 that will allow for reliability, persistence, security and extensibility.

116 The ebXML Message Service is provided for environments requiring a robust, yet low cost  
 117 solution to enable electronic business. It is one of the three "infrastructure" components of ebXML  
 118 that includes: Registry/Repository [ebXMLRegRep], Collaboration Protocol Profile/Agreement  
 119 [ebXMLTP] and the ebXML Message Service.

120 **6.2 Message Service Overview**

121 The *ebXML Messaging Service* may be conceptually broken down into following three parts: (1)  
 122 an abstract *Service Interface*, (2) functions provided by the *Messaging Service Layer*, and (3) the  
 123 mapping to underlying transport service(s).

124 The following diagram depicts a logical arrangement of the functional modules that exist within  
 125 the ebXML *Messaging Services* architecture. These modules are arranged in a manner to  
 126 indicate their inter-relationships and dependencies.



127

128 **Figure 6-1 Typical Relationship between ebXML MSH Components**

129 <DB>Diagram needs to be simplified and ~~A~~an explanation of these components ~~is needed~~needs  
130 to be provided. (Ralph & Chris)</DB>

## 131 7 Packaging Specification

### 132 7.1 Introduction

133 An ebXML Message consists of:

- 134 • an outer Communication Protocol Envelope, such as HTTP or SMTP,
- 135 • an inner communication “protocol independent” ebXML Message Envelope, specified
- 136 using MIME multipart/related, that contains the two main parts of the Message:
  - 137 - an ebXML Header Container that is used to envelope one ebXML Header Document,
  - 138 - an optional, single ebXML Payload Container that MUST be used to envelope the
  - 139 actual payload (transferred data) of the Message Communication Protocol Envelope
  - 140 (SMTP, HTTP, etc)

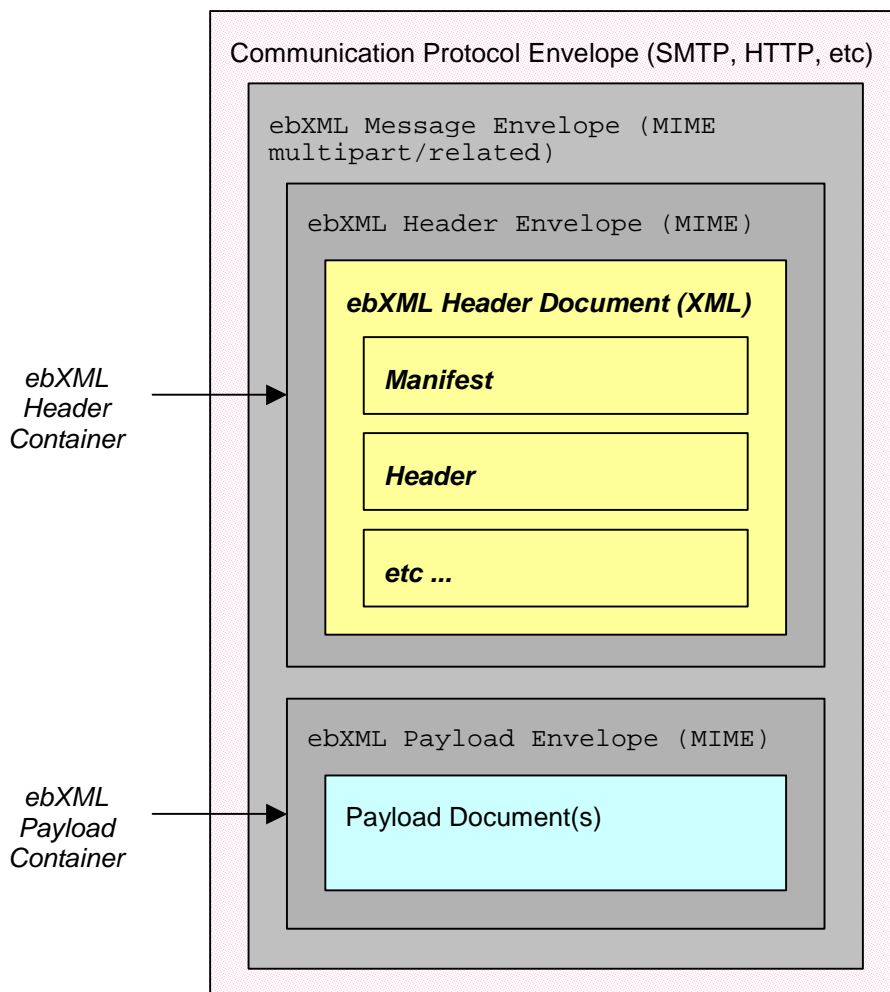


Figure 7-1 ebXML

141  
142

#### Message Structure

### 143 7.1.1 ebXML Header Envelope and ebXML Payload Envelope

144 An ebXML Header Envelope and an ebXML Payload Envelope are constructed of standard,  
145 MIME components.

146 The ebXML Header Envelope contains a single ebXMLHeader document (see section 8). The  
 147 ebXML Payload Envelope can contain any electronic data that can be transported within MIME.  
 148 Any special considerations for the usage of the ebXML Message Envelope in HTTP and SMTP  
 149 transports are described in Appendix C.

## 150 7.1.2 MIME usage Conventions

151 Values associated with MIME header attributes are valid in both quoted and unquoted form. For  
 152 example, the forms `type="ebxml"` and `type=ebxml` are both valid.

## 153 7.2 ebXML Message Envelope

154 The MIME structured *ebXML Message Envelope* is used to identify the message as an ebXML  
 155 compliant structure and encapsulates the header and payload in MIME body parts. It MUST  
 156 conform to [RFC2045] and MUST contain a Content-Type MIME header.

### 157 7.2.1 Content-Type

158 The MIME Content-Type MUST be set to `multipart/related` for all *ebXML Message*  
 159 *Envelopes*. For example:

```
160 Content-Type: multipart/related;
```

162 The MIME Content-Type header contains three attributes:

- 163 • type
- 164 • boundary
- 165 • version

#### 166 7.2.1.1 type Attribute

167 The MIME `type` attribute is used to identify the *ebXML Message Envelope* as an ebXML  
 168 compliant structure. It conforms to a MIME XML Media Type [XMLMedia] and MUST be set to  
 169 `"application/vnd.eb+xml"`. This media type is derived from the `application/xml` type  
 170 and shares many semantics with that type. To that type, `application/vnd.eb+xml` adds a  
 171 specific application context, the ebXML Message Service. For example:

```
172 type="application/vnd.eb+xml"
```

#### 174 7.2.1.2 boundary Attribute

175 The MIME boundary attribute is used to identify the body part separator used to identify the start  
 176 and end points of each body part contained in the message. The MIME boundary SHOULD be  
 177 chosen carefully in order to ensure that it does not occur within the content area of a body part  
 178 see [RFC 2045] for guidance on how to do this. For example:

```
179 boundary:="-----8760"
```

#### 181 7.2.1.3 version Attribute

182 [The MIME `version` attribute indicates the version of the ebXML Message Service Specification](#)  
 183 [to which the \*ebXML Message Envelope\* conforms. All message headers SHOULD USE "0.92".](#)  
 184 [For example:](#)

185 ~~The MIME `version` attribute is used to identify the particular version of ebXML Message Envelope~~  
 186 ~~being used. All message headers SHOULD USE "0.91". "0.92". For example:~~

```
187 version="0.91" version="0.92"
```

## 189 7.2.2 ebXML Message Envelope Example

190 An example of a compliant *ebXML Message Envelope* header appears as follows:

191  
192

```
Content-Type: multipart/related; type="application/vnd.eb+xml"; boundary:="-----8760";
```

## 193 7.3 ebXML Header Container

194 The *ebXML Header Container* is a MIME body part that MUST consist of:

- 195 • one *ebXML Header Envelope*, that contains
- 196 • one XML *ebXML Header document* (see section 8).

197 The following rules apply:

- 198 • the *ebXML Header Container* MUST be the first MIME body part in the *ebXML Message*.
- 199 • there MUST be one and only one *ebXML Header Document* in each *ebXML Message*.

200 Note that, an *ebXML Payload Container* may be a completely encapsulated *ebXML Message*.

201 The MIME based *ebXML Header Envelope* conforms to [RFC 2045] and MUST consist of the  
202 following MIME headers:

- 203 • Content-ID
- 204 • Content-Type

### 205 7.3.1 Content-ID

206 The `Content-ID` MIME header identifies this instance of an ebXML Message header body part.  
207 The value for `Content-ID` SHOULD be a unique identifier, in accordance with RFC 2045. For  
208 example:

209  
210

```
Content-ID: <2000-0722-161201-123456789@ebxmlhost.realm>
```

### 211 7.3.2 Content-Type

212 The MIME `Content-Type` for an ebXML header is identified with the value  
213 "application/vnd.eb+xml". `Content-Type` contains two attributes:

- 214 • version
- 215 • charset

#### 216 7.3.2.1 version Attribute

217 [The MIME `version` attribute indicates the version of the ebXML Message Service Specification](#)  
218 [to which the \*ebXML Header Envelope\* and \*ebXML Header Document\* conform. All message](#)  
219 [headers MUST USE "0.92". Future versions of this specification may require other values of this](#)  
220 [attribute. However, the value specified here MUST match that specified in the `version` attribute](#)  
221 [of the \*ebXML Header Document\* for all versions of this specification. For example:](#)

222 ~~The MIME `version` attribute indicates the version of the ebXML Message Service Specification~~  
223 ~~to which the *ebXML Header Document* conforms. For example:~~

224  
225

```
version="0.91";version="0.92";
```

#### 226 7.3.2.2 charset Attribute

227 The MIME `charset` attribute identifies the character set used to create the *ebXML Header*  
228 *Document*. The semantics of this attribute are described in the "charset parameter / encoding  
229 considerations" of `application/xml` as specified in [XML/Media]. The list of valid values can  
230 be found at <http://www.iana.org/>.

231 If both are present, the MIME `charset` attribute SHALL be equivalent to the encoding  
 232 declaration of the ebXML Header Document (see section 8). If provided, the MIME `charset`  
 233 attribute MUST NOT contain a value conflicting with the encoding used when creating the ebXML  
 234 Header Document. For maximum interoperability it is RECOMMENDED that [UTF-8] be used  
 235 when encoding this document. Due to the processing rules defined for media types derived from  
 236 `application/xml` [XMLMedia], this MIME attribute has no default. For example:

```
237 charset="UTF-8"
```

239 **7.3.3 ebXML Header Envelope Example**

240 The following represents an example of an *ebXML Header Envelope* and *ebXML Header*  
 241 *Document*:

242	Content-ID: ebxmlheader-123@ebxmlhost.realm --	MIME ebXML	
243	Content-Type: application/vnd.eb+xml;	Header Envelope	
244	<del>version="0.91";</del> version="0.92"; charset="UTF-8"	--	ebXML
245	<ebXMLHeader>		Header
246	-----		Container
247	<Manifest>.....	XML ebXML Header	
248	</Manifest>	Document	
249	<Header>.....		
250	</Header>		
251	<Routing Header>.....		
252	</Routing Header>		
253	</ebXMLHeader>		
254	-----		

255 A complete example of an *ebXML Header Container* is presented in Appendix B. That example  
 256 includes the `charset` attribute and portions of an *XML Prolog* (see sect 8.1), none of which is  
 257 required to appear in an *ebXML Header Container* or *ebXML Header Document*. Appendix B  
 258 also includes the outer *ebXML Message Envelope* and a complete (valid) **ebXMLHeader** element  
 259 rather than the outline shown above.

260 **7.4 ebXML Payload Container**

261 If the *ebXML Message* contains a payload, then a single *ebXML Payload Container* MUST be  
 262 used to envelop it.

263 If there is no payload within the *ebXML Message* then the *ebXML Payload Container* MUST not  
 264 be present.

265 The contents of the *ebXML Payload Container* MUST be identified by the *Message Manifest*  
 266 element within the *ebXML Header Document* (see section 8.3).

267 If the *Message Manifest* is an empty XML element, the *ebXML Payload Container* MUST NOT be  
 268 present in the *ebXML Message*.

269 If an *ebXML Payload Container* is present, it MUST conform to MIME [RFC2045] and MUST  
 270 consist of:

- 271 • a MIME header portion - the *ebXML Payload Envelope*, and
- 272 • a content portion - the payload itself that may be of any valid MIME type.

273 The *ebXML MIME Payload Envelope*, MUST consist of the following MIME headers:

- 274 • Content-ID
- 275 • Content-Type

276 The *ebXML Message Service Specification* makes no provision, nor limits in any way the  
 277 structure or content of payloads. Payloads MAY be a simple-plain-text-object or complex nested  
 278 multipart objects. This is the implementer's decision.

279 **7.4.1 Content-ID**

280 The `Content-ID` MIME Header is used to uniquely identify an instance of an *ebXML Message*  
 281 payload body part. The value for `Content-ID` SHOULD be a unique identifier, in accordance  
 282 with MIME [RFC 2045]. For example:

```
283 Content-ID: <2000-0722-161201-123456789@ebxmlhost.realm>
```

285 **7.4.2 Content-Type**

286 The MIME `Content-Type` for an ebXML payload is determined by the implementer and is used  
 287 to identify the type of data contained in the content portion of the *ebXML Payload Container*. The  
 288 MIME `Content-Type` MUST conform to [RFC2045]. For example:

```
289 Content-Type: application/xml
```

291 **7.4.3 Example of an ebXML MIME Payload Container**

292 The following represents an example of an *ebXML MIME Payload Envelope* and a payload:

```
293 Content-ID: ebxmlpayload-123@ebxmlhost.realm -- |
294 ebXML MIME |
295 Content-Type: application/xml -----| Payload Envelope | ebXML
296 | | | | Payload
297 <Invoice> -----| | Container
298 <Invoicedata>..... | Payload
299 </Invoicedata> |
300 </Invoice> -----| |
```

301 A complete example of the ebXML Payload Container is presented in Appendix XX.

302 **7.5 Additional MIME Parameters**

303 ~~The Any~~ MIME part described by this specification ~~may~~**MAY** contain additional MIME  
 304 parameters in conformance with the [MIME] specification. Implementations MAY ignore any  
 305 MIME parameter not defined in this specification. Implementations MUST ignore any MIME  
 306 parameter that they do not recognize.

307 For example, an implementation could include content-length in a message. However, a  
 308 recipient of a message with content-length could ignore it.

309 ~~**-If these are present then they MUST be ignored.**~~

310 Reporting MIME Errors

311 **7.6 Reporting MIME Errors**

312 If a MIME error is detected in the *ebXML Header Envelope* or the *ebXML Payload Envelope* then  
 313 it MUST be reported by sending an ebXML message containing an **ebXMLHeader** element with  
 314 an **ErrorList** element (see section 8.8) where **errorCode** is set to **MimeProblem** and a **severity**  
 315 set to **Error**. See section 11 for more details on how to indicate an error.

## 316 8 ebXML Header Document

317 The ebXML Header Document is a single [XML] document with a number of principal header-  
318 elements. In general, separate principal-header elements are used where:

- 319 • different software components are likely to be used to generate that header-element,
- 320 • the element is not always present,
- 321 • the structure of the header element might vary independently of the other header-  
322 elements, or
- 323 • the data contained in the header-element MAY need to be digitally signed separately  
324 from the other header-elements.

### 325 8.1 XML Prolog

326 The ebXML Header Document's XML Prolog MAY contain an XML declaration or a document  
327 type declaration. This specification has defined no additional comments or processing  
328 instructions that may appear in the XML prolog. For example:

```
329 <?xml version="1.0" encoding="UTF-8"?>
330 <!DOCTYPE ebXMLHeader SYSTEM "level1-10122000.dtd">
331 <ebXMLHeader>...</ebXMLHeader>
332
```

#### 333 8.1.1 XML Declaration

334 The XML declaration MAY be present in an ebXML Header Document. If present, it MUST  
335 contain the version specification required by the XML Recommendation [XML]: version='1.0' and  
336 MAY contain an encoding declaration and standalone document declaration. The semantics  
337 described below MUST be implemented by a compliant ebXML Message Service.

#### 338 8.1.2 Encoding Declaration

339 *<DB>This section isn't clear to me. I really could not work out what is or is not valid and what do  
340 you do if you get an inconsistency between the XML prolog and the MIME header. e.g. do you  
341 ignore it, assume a value or report an error.</DB>*

342 If both *<DB>the encoding declaration and the MIME charset?</DB>* are present, the XML prolog  
343 for the ebXML Header Document SHALL contain the encoding declaration that SHALL be  
344 equivalent to the `charset` attribute of the MIME Content-Type of the ebXML Message Header  
345 Container (see section 7.3).

346 If provided, the encoding declaration MUST NOT contain a value conflicting with the encoding  
347 used when creating the ebXML Header Document. It is RECOMMENDED that UTF-8 be used  
348 when encoding the ebXML Header Document.

349 If the character encoding cannot be determined by an XML processor using the rules specified in  
350 section 4.3.3 of [XML], the XML declaration and its contained encoding declaration SHALL be  
351 provided in the ebXML Header Document.

352 NOTE: The encoding declaration is not required in an XML document according to the XML  
353 version 1.0 specification [XML].

354 For example:

```
355 <?xml version="1.0" encoding="UTF-8"?>
```

#### 356 8.1.3 Standalone Document Declaration

357 The standalone document declaration, if present, MAY appear as ***standalone='yes'*** if and only if  
358 all of the validity requirements specified in section 2.9 of the XML Recommendation [XML] are  
359 met. It is RECOMMENDED that ebXML Header Documents omit this declaration.

360 *<DB>What do you do if the XML Recommendation is not met?</DB>*

### 361 8.1.4 Document Type Declaration

362 When the *ebXML Header Document* will or may be processed by an XML processor not  
 363 ~~complaintcompliant~~ with the XML Schema Recommendation [XMLSchema], a document type  
 364 declaration containing a SYSTEM identifier of "level1-10122000.dtd" MUST be included. For  
 365 example:

```
366 <!DOCTYPE ebXMLHeader SYSTEM "level1-10122000.dtd">
```

367  
 368 ~~<DB>Looks to me like we should mandate its use<DB>1. Do we need to mandate use of a~~  
 369 ~~DOCTYPE and then maybe remove it in a later version of the spec when everyone is using XML~~  
 370 ~~Schema.</DB>~~  
 371 ~~2. Do we we need to be prescriptive about what goes in the Prolog depending on whether we~~  
 372 ~~have a DTD or XSD.(Saikat)</DB>~~

## 373 8.2 ebXMLHeader Element

374 The root element of the *ebXML Header Document* is named the **ebXMLHeader**. Its structure is  
 375 described below.

### 376 8.2.1 ebXMLHeader attributes

377 There are two attributes ~~associated defined for with~~ the **ebXMLHeader** element, they are as  
 378 follows:

- 379 • Namespace (xmlns)
- 380 • Version

381 [Additional namespace declarations and namespace-qualified attributes from foreign namespaces](#)  
 382 [MAY be added to support extensions to the \*\*ebXMLHeader\*\* document.](#)

#### 383 8.2.1.1 Namespace attribute

384 The namespace declaration (xmlns) (see [XML Namespace]) has a REQUIRED value of  
 385 "http://www.ebxml.org/namespaces/messageHeader".

#### 386 8.2.1.2 version attribute

387 [The required \*\*version\*\* attribute indicates the version of the ebXML Message Service Specification](#)  
 388 [to which the \*ebXML Header Document\* conforms. Its purpose is to provide for future versioning](#)  
 389 [capabilities. All \*ebXML Header Documents\* MUST USE "0.92". Future versions of this](#)  
 390 [specification SHALL require other values of this attribute. However, the value specified here](#)  
 391 [MUST match that specified in the MIME \*\*version\*\* attribute of the \*ebXML Header Envelope\* for all](#)  
 392 [versions of this specification.](#)

393 ~~The **version** attribute is required. Its purpose is to provide for future versioning capabilities. It has~~  
 394 ~~a default value of '0.91'. '0.92'.~~

### 395 8.2.2 ebXMLHeader elements

396 An ebXML Header Document consists of the following principal header elements:

- 397 • **Manifest** – an element that points to any data present either in the *ebXML Payload*  
 398 *Container* or elsewhere, e.g. on the web
- 399 • **Header** – a REQUIRED element that contains routing information for the message  
 400 (To/From, etc.) as well as other context information about the message
- 401 • **RoutingHeaderList** – an element that contains entries that identify the Message Service  
 402 Handler (MSH) that sent and should receive the message. This element can be omitted.
- 403 • **ApplicationHeaders** – an element that can be used by a process or service to include  
 404 additional information that needs to be associated with the data in the *ebXML Payload*  
 405 but is not contained within it

- 406 • **StatusData** – an element that is used by a MSH when responding to a request on the
- 407 status of a message that was previously received
- 408 • **ErrorList** – an element that contains a list of the errors that have been found in a
- 409 message
- 410 • **Acknowledgment** – an element that is used by a MSH to indicate that a message has
- 411 been received
- 412 • **Signature** – an element that contains a digital signature that conforms to [XMLDSIG] that
- 413 signs data associated with the message
- 414 • **#wildcard** - any namespace-qualified element content belonging to a foreign namespace

### 415 8.2.3 Combining Principal Header Elements

416 This section describes how the various principal header elements may be used in combination.

#### 417 8.2.3.1 Manifest element

418 The **Manifest** element MUST be present if there is any data associated with the message that is  
 419 not present in the *ebXML Header Document*. This applies specifically to data in the *ebXML*  
 420 *Payload Container* or elsewhere, e.g. on the web.

#### 421 8.2.3.2 Header element

422 The **Header** element MUST be present in every message.

#### 423 8.2.3.3 RoutingHeaderList element

424 The **RoutingHeaderList** element MAY be present in any message. It MUST be present if the  
 425 message is being sent reliably (see section 10) [or over multiple hops \(see section 8.5.3\)](#).

#### 426 8.2.3.4 ApplicationHeaders element

427 The **ApplicationHeaders** element MAY be present on any message except a message that  
 428 contains ~~one or more of the following:~~

- 429 – an **ErrorList** element with a **highestSeverity** attribute set to **Error**.
- 430 – ~~a **StatusData** element~~

#### 431 8.2.3.5 StatusData element

432 This element MUST NOT be present with the following elements:

- 433 • a **Manifest** element
- 434 – ~~an **ApplicationHeaders** element,~~
- 435 • an **ErrorList** element with a **highestSeverity** attribute set to **Error**

#### 436 8.2.3.6 ErrorList element

437 If the **highestSeverity** attribute on the **ErrorList** is set to **Warning**, then this element MAY be  
 438 present with any other element.

439 If the **highestSeverity** attribute on the **ErrorList** is set to **Error**, then this element MUST NOT be  
 440 present with the following:

- 441 • a **Manifest** element
- 442 • an **ApplicationHeaders** element
- 443 • a **StatusData** element

444 **8.2.3.7 Acknowledgment element**

445 An **Acknowledgment** element MAY be present on any message.

446 **8.2.3.8 Signature element**

447 A **Signature** element MAY be present on any message.

448 **8.2.3.9 #wildcard element content**

449 Any namespace-qualified element content MAY be added to provide for the extensibility of the  
 450 ebXMLHeader. Extension element content MUST be namespace-qualified in accordance with  
 451 [XMLNamespaces] and MUST belong to a foreign namespace. A foreign namespace is one that  
 452 is NOT <http://www.ebxml.org/namespaces/messageHeader>.

453 Any namespace-qualified element added SHOULD include the global **mustUnderstand** attribute.  
 454 If the **mustUnderstand** attribute is NOT present, the default value implied is 'false'. If an  
 455 implementation of the MSH does not recognize the namespace of the element and the value of  
 456 the **mustUnderstand** attribute is 'true' then the MSH SHALL respond with a message that  
 457 includes an **errorCode** of **NotSupported** in an **Error** element as defined in section 8.8. If the  
 458 value of the **mustUnderstand** attribute is 'false' or if the **mustUnderstand** attribute is not present  
 459 then an implementation of the MSH MAY ignore the namespace-qualified element and its  
 460 content.

462 **8.2.4 ebXMLHeader sample**

463 The following is a sample **ebXMLHeader** document fragment demonstrating the overall structure:

```
464 <?xml version="1.0" encoding="UTF-8"?>
465 <ebXMLHeader xmlns="http://www.ebxml.org/namespaces/messageHeader" Version="0.91"Version="0.92"
466 >
467 <Manifest>...</Manifest>
468 <Header>...</Header>
469 <RoutingHeaderList>...</RoutingHeaderList>
470 </ebXMLHeader>
```

472 **8.3 Manifest element**

473 The **Manifest** element is a composite element consisting of one or more **Reference** elements.  
 474 Each **Reference** element identifies data associated with the message, whether included as part  
 475 of the message as payload document(s) contained in the **ebXML Message Container**, or remote  
 476 resources accessible via a URL. The **Manifest** element, if present, SHALL be the first child  
 477 element of the **ebXMLHeader**. ~~It identifies the payload document(s) contained in the **ebXML**~~  
 478 ~~**Message Container**.~~The purpose of the **Manifest** is as follows:

- 479 • to make it easier to directly extract a particular document associated with this **Message**.
- 480 • to enable a MSH to check the integrity of a **Message**
- 481 • to allow an application to determine whether it can process the payload without having to  
 482 parse it.

483 The **Manifest** element ~~MAY~~**MUST** have a single attribute: **id** that is an XML ID.

484 **8.3.1 Reference element**

485 The **Reference** element is a composite element consisting of the following subordinate elements:

- 486 ~~—**Description**— a textual description of the payload object referenced by the parent~~  
 487 ~~**Reference** element~~
- 488 • **Schema** - information about the schema(s) that define s the instance document identified  
 489 in the parent **Reference** element

- 490 • Description - a textual description of the payload object referenced by the parent
- 491 Reference element
- 492 • **#wildcard** - any namespace-qualified element content belonging to a foreign namespace

493 The **Reference** element itself is an [XLINK] simple link. XLINK is presently a Candidate  
 494 Recommendation (CR) of the W3C. It should be noted that the use of XLINK in this context is  
 495 chosen solely for the purpose of providing a concise vocabulary for describing an association.  
 496 Use of an XLINK processor or engine is NOT REQUIRED, but MAY prove useful in certain  
 497 implementations.

498 The **Reference** element has the following attribute content in addition to the element content  
 499 described above:

- 500 • **id** - an ~~optional~~ **REQUIRED** XML ID for the **Reference** element.
- 501 • **xlink:type** - this ~~REQUIRED~~ attribute defines the element as being an XLINK simple link.  
 502 It has a fixed value of 'simple'.
- 503 • **xlink:href** - this **REQUIRED** attribute has a value that is the URI of the payload object  
 504 referenced. It SHALL conform to the [XLINK] specification criteria for a simple link.
- 505 • **xlink:role** - this ~~REQUIRED~~ attribute identifies ~~the role that the payload object~~  
 506 ~~referenced serves some resource that describes the payload object or its purpose. If~~  
 507 ~~present, then it MUST SHALL~~ have a value that is a valid URI in accordance with the  
 508 [XLINK] specification.
- 509 ~~**xlink:label** - this attribute MAY be present and SHALL be used in accordance with the~~  
 510 ~~[XLINK] specification.~~
- 511 • Any other namespace-qualified attribute MAY be present. A receiving MSH MAY choose  
 512 to ignore any foreign namespace attributes other than those defined above.

513 **8.3.1.1 Description element**

514 ~~The **Description** is an OPTIONAL textual description of the payload object referenced by the~~  
 515 ~~parent **Reference** element. The language of the description is defined by a required **xml:lang**~~  
 516 ~~attribute. The **xml:lang** attribute MUST comply with the rules for identifying languages specified~~  
 517 ~~in [XML]. This element is provided solely for the purpose of providing a human readable~~  
 518 ~~description of the payload object identified by the parent **Reference** element.~~

519 **8.3.1.28.3.1.1 Schema element**

520 ~~If the item being referenced has schema(s) of some kind that describe it (e.g. an XML Schema,~~  
 521 ~~DTD or a database schema), then ~~T~~the **Schema** element ~~MAY SHOULD~~ be present as a child of  
 522 the **Reference** element. ~~It provides a means of identifying the schema, and its version, that~~  
 523 ~~defines the payload object identified by the parent **Reference** element. ~~It has no element or text~~~~  
 524 ~~content.~~The Schema element contains the following attributes:~~

- 525 • location - the **REQUIRED** URI of the schema
- 526 • version - a version identifier of the schema
- 527 ~~location - the URI of the schema~~

528 **8.3.1.2 Description element**

529 The **Reference** element MAY contain zero or more **Description** elements. The **Description** is a  
 530 textual description of the payload object referenced by the parent **Reference** element. The  
 531 language of the description is defined by a **REQUIRED xml:lang** attribute. The **xml:lang** attribute  
 532 MUST comply with the rules for identifying languages specified in [XML]. This element is provided  
 533 solely for the purpose of to allow providing a human readable description of the payload object  
 534 identified by the parent **Reference** element. If multiple **Description** elements are present, each  
 535 SHOULD have a unique **xml:lang** attribute value. An example of a **Description** element follows.

536 `<Description xml:lang="en-gb">Purchase Order for 100,000 widgets</Description>`

537 [8.3.1.3 #wildcard element](#)538 [Refer to section 8.2.3.9 for discussion of #wildcard element handling.](#)539 **8.3.2 What References are Included in a Manifest**540 The designer of the protocol or application that is using ebXML Messaging decides what payload  
541 data is referenced by the Manifest and the values to be used for *xlink:role*.542 **8.3.3 Manifest Validation**543 If an *xlink:href* attribute contains a URI that is a content id (URI scheme "cid") then a MIME  
544 part with that content-id MUST be present in the ebXML Payload Container of the message.  
545 If it is not, then the error SHALL be reported to the From Party with an *errorCode* of  
546 *MimeProblem* and a *severity* of *Error*.547 If an *xlink:href* attribute contains a URI that is not a content id (URI scheme "cid") and that URI  
548 cannot be resolved, then it is an implementation decision on whether to report the error. If the  
549 error is to be reported, then it SHALL be reported to the From Party with an *errorCode* of  
550 *MimeProblem* and a *severity* of *Error*.551 **8.3.28.3.4 Manifest sample**552 The following fragment demonstrates a typical *Manifest* for a message with a single payload  
553 MIME body part:554  
555 

```
<Manifest id="Manifest">
556   <Reference id="pay01"
557     xlink:href="cid:payload-1" xlink:label="PO"
558     xlink:role="http://regrep.org/gci/purchaseOrder">
559     <Description>Purchase Order for 100,000 widgets</Description>
560     <Schema location="http://regrep.org/gci/purchaseOrder/po.xsd"
561       version="1.0"/>
562   </Reference>
563 </Manifest>
```

564 **8.4 Header element**565 The *Header* element immediately follows the *Manifest* element. It is REQUIRED in all  
566 *ebXMLHeader* documents. The *Header* element is a composite element comprised of the  
567 following subordinate elements:

- 568
- *From*
  - *To*
  - *CPAId*
  - *ConversationId*
  - *Service*
  - *Action*
  - *MessageData*
  - *ReliableMessagingInfoQualityOfServiceInfo*
  - *SequenceNumber*
  - *Description*
  - *#wildcard*

579 The Header attribute MAY have an attribute: *id* that is of type XML ID.580 **8.4.1 From and To elements**581 The REQUIRED *From* element identifies the *Party* that originated the message. ~~The *From*~~  
582 ~~element consists of a *PartyId* element.~~

583 The **REQUIRED To** element identifies *Party that is* the intended recipient of the message. ~~As with~~  
 584 ~~Both To and From, it is a can be~~ logical identifiers such as a DUNS number or identifiers that  
 585 ~~also imply a physical location, such as an email address, that is comprised of a PartyId element.~~

586 The ~~PartyId-From and the To~~ elements ~~has have~~ a single child element, PartyId.

587 The PartyId element has a single attribute, type and content that is a string value.

588 ~~If the type attribute is present, then it MUST be a URN. It indicates the domain of names to~~  
 589 ~~which the string, in the content of the From or To element, belongs, that the parties that are~~  
 590 ~~sending and receiving the message know, by some other means, how to interpret the content of~~  
 591 ~~the PartyId element. The two parties MAY use the value of the type attribute to assist in the~~  
 592 ~~interpretation.~~

593 If the **PartyId type** attribute is not present, the content of the ~~PartyIdFrom or To~~ **PartyId**  
 594 elements ~~is~~ MUST be an URI [RFC 2396] otherwise report an error (see section 11) with **errorCode**  
 595 set to **Inconsistent** and **severity** set to **error**. It is strongly RECOMMENDED that the content  
 596 is be an URN.

597 The following fragment demonstrates usage of the **From** and **To** elements. The first illustrates a  
 598 user-defined numbering scheme, and the second a URN.

599  
 600  
 601  
 602  
 603  
 604  
 605  
 606  
 607  
 608  
 609

```
<From>
  <PartyId>
  <PartyId type="MyNumberingSchemeurn:duns.com">1234567890123</PartyId>
</PartyId>
</From>
<To>
  <PartyId><PartyId">urn:dnb.com:duns:3210987654321smtp:joe@example.com</PartyId>
</PartyId>
</To>
```

610 **8.4.2 CPAId element**

611 The **REQUIRED CPAId element** is a string ~~that that~~ identifies the *Collaboration Protocol*  
 612 *Agreement (CPA)* that governs the processing of the message. ~~The CPAId MAY be a URI,~~  
 613 ~~possibly established by registering a CPA with an ebXML compliant Registry, that identifies the~~  
 614 ~~CPA uniquely. It MUST be an identifier that is unique within the combination of the From and To~~  
 615 ~~Parties.~~

616 A Party that receives the message, must be able to resolve the CPAId to the CPA instance as  
 617 information in the CPA is used, for example, by Reliable Messaging (see section 10). It is  
 618 therefore RECOMMENDED that the CPAId is a URI.

619 <DB>The usage of CPAIds is not resolved. There are issues around using CPAs in the area of  
 620 multiple hops and requiring use of URIs (or URLs). </DB>

621

622 **8.4.3 ConversationId element**

623 The **REQUIRED ConversationId element** is a string that identifies the set of related messages  
 624 that make up a conversation between two **Parties**. The **Party** that initiates a conversation  
 625 determines the value of the **ConversationId** element that shall be reflected in all messages  
 626 pertaining to that conversation.

627 The ConversationId enables the recipient of a message to identify the instance of an application  
 628 or process that generated or handled earlier messages within a conversation. It remains constant  
 629 for all messages within a conversation.

630 The value used for a ConversationId is implementation dependent.

631 Note that implementations are free to choose how they will identify and store conversational state  
 632 related to a specific **ConversationId**. Implementations SHOULD provide a facility for mapping

633 between their identification schema and a ConversationId generated by another implementation.  
634 *<DB>I think that this last sentence should be deleted as it requiring implementation behavior that*  
635 *is an implementation decision.</DB>*

#### 636 8.4.4 Service element

637 The **REQUIRED Service** element identifies the service that ~~SHOULD~~ acts on the ~~payload in the~~  
638 message. It is specified by the designer of the service. The designer of the service may be:

- 639 • a standards organization, or
- 640 • an individual or enterprise

641 *Note that in the context of an ebXML Business Process model, a Service element identifies a*  
642 *Business Transaction. <DB>This definition needs to go in the glossary</DB>*

643 The **Service** element has a single **type** attribute.

644

##### 645 8.4.4.1 type attribute

646 If the **type** attribute is present, then it indicates that the parties that are sending and receiving the  
647 message know, by some other means, how to interpret the content of the **Service** element. The  
648 two parties MAY use the value of the **type** attribute to assist in the interpretation.

649 If the **type** attribute is not present, the content of the **Service** element MUST be a URI [RFC  
650 2396]. ~~otherwise if it is not a URI then there is an error report an error with an errorCode of~~  
651 *Inconsistent and a Severity of Error (see section 11).*

##### 652 8.4.4.2 ebXML Message Service ~~Header~~ namespace

653 URIs in the **Service** element that start with the namespace:

654 *http://www.ebxml.org/namespaces/messageService* are reserved for use by this specification.

#### 655 8.4.5 Action element

656 The **REQUIRED Action** element identifies a process within a ~~Service, that Service that~~  
657 processes the Message. **Action** SHALL be unique within the **Service** in which it is defined.

#### 658 8.4.6 MessageData element

659 The **REQUIRED MessageData** element provides a means of uniquely identifying an *ebXML*  
660 *Message*. It contains the following three elements:

- 661 • **MessageId**
- 662 • **Timestamp**
- 663 • **RefToMessageId**

##### 664 8.4.6.1 MessageId element

665 The **REQUIRED element MessageId** is a unique identifier for the message conforming to  
666 [RFC2392]. The "local part" of the identifier as defined in [RFC2392] is implementation  
667 dependent.

##### 668 8.4.6.2 Timestamp element

669 The **Timestamp** is a value representing the time that the message header was created  
670 conforming to [ISO-8601]. The format of CCYYMMDDTHHMMSS.SSSZ is **REQUIRED** to be  
671 used. This time format is Coordinated Universal Time (UTC).

672 *<DB>Should we make this compliant with an XML Schema timeInstant instead? </DB>*

673 **8.4.6.3 RefToMessageId element**

674 The **RefToMessageId** element has a cardinality of zero or one. When present, it MUST contain  
 675 the **MessageId value** of an earlier ebXML Message to which this message relates. If there is no  
 676 earlier related message, the element MUST NOT be present.

677 For Error messages, the **RefToMessageId** element is REQUIRED and its value MUST be the  
 678 **MessageId** value of the *message in error* (as defined in section 8.8).

679 For Acknowledgment Messages, the **RefToMessageId** element is REQUIRED, and its value  
 680 MUST be the **MessageId value** of the ebXML Message being acknowledged. See also sections  
 681 8.2.3.7 and 10.

682 **8.4.7 ReliableMessagingInfoQualityOfServiceInfo element**

683 The **ReliableMessagingInfoQualityOfServiceInfo** element identifies the quality of service with  
 684 which the message ~~MUST is be~~ delivered. This element has ~~a~~ four attributes:

- 685 • **deliverySemantics**
- 686 • **messageOrderSemantics**
- 687 • **deliveryReceiptRequested**
- 688 • **syncReplyMode**, and
- 689 • **timeToLive**.

690 The **QualityOfServiceInfo** element is present if any of the attributes within the element need to  
 691 be set to their non-default value.

692 **8.4.7.1 deliverySemantics attribute**

693 The **deliverySemantics** ~~element attribute~~, if present, over-rides the value of the same parameter  
 694 in the CPA. If it is not present, the value in the CPA MUST be used.

695 The **deliverySemantics** parameter/element MUST used by the *From Party* MSH to indicate  
 696 whether the Message must be sent reliably. Valid Values are:

- 697 • **OnceAndOnlyOnce**. The message must be sent using a **reliableMessagingMethod**  
 698 that will result in the application or other process at the *To Party* receiving the message  
 699 once and only once
- 700 • **BestEffort** The reliable delivery semantics are not specified. In this case the value of  
 701 **reliableMessagingMethod** is ignored.

702 The default value for **deliverySemantics** is specified in the CPA. If no value is specified in the  
 703 CPA then the default value is **BestEffort**.

704 If **deliverySemantics** is set to **OnceAndOnlyOnce** then the *From Party* MSH and the *To Party*  
 705 MSH must adopt the Reliable Messaging behavior (see section 10) that describes how messages  
 706 are resent in the case of failure and duplicates are ignored.

707 If **deliverySemantics** is set to **BestEffort** then a MSH that received a message that it is unable  
 708 to deliver MUST NOT take any action to recover or otherwise notify anyone of the problem, and  
 709 the MSH that sent the message must not attempt to recover from any failure.

710 This means that duplicate messages might be delivered to an application and persistent storage  
 711 of messages is not required.

712 If the *To Party* is unable to support the type of Delivery Semantics requested, then the *To Party*  
 713 SHOULD report the error to the *From Party* using an **ErrorCode** of **NotSupported** and a  
 714 **Severity** of **Error**.

```
715 <ReliableMessagingInfoQualityOfServiceInfo deliverySemantics="OnceAndOnlyOnce" />
```

717 **8.4.7.1 messageOrderSemantics attribute**

718 The **messageOrderSemantics** attribute, if present, over-rides the value of the same parameter  
719 in the CPA. If it is not present, the value in the CPA MUST be used.

720 The **messageOrderSemantics** parameter/attribute MUST be used by the *From Party* MSH to  
721 indicate whether the Message is passed to the receiving application in the order which the  
722 sending application specified. Valid Values are:

- 723 • **Guaranteed**. The messages are passed to the receiving application in the order which  
724 the sending application specified.
- 725 • **NotGuaranteed** The messages may be passed to the receiving application in different  
726 order from the order which sending application specified.

727 The default value for **messageOrderSemantics** is specified in the CPA. If no value is specified in  
728 the CPA then the default value is **NotGuaranteed**.

729 If **messageOrderSemantics** is set to **Guaranteed** then the *To Party* MSH MUST correct invalid  
730 order of messages using the value of **SequenceNumber** in the conversation specified the  
731 **ConversationId**. The **Guaranteed** semantics can be set only when **deliverySemantics** is  
732 **OnceAndOnlyOnce**. If **deliverySemantics** is not **OnceAndOnlyOnce** then report the error to  
733 the *From Party* with an **errorCode** of **Inconsistent** and a **severity** of **Error** (see section **Error!**  
734 **Reference source not found.14**).

735 If **deliverySemantics** is set to **NotGuaranteed**, then the *To Party* MSH does not need to correct  
736 invalid order of messages. If the *To Party* is unable to support the type of  
737 **MessageOrderSemantics** requested, then the *To Party* MUST report the error to the *From Party*  
738 using an **ErrorCode** of **NotSupported** and a **Severity** of **Error**. A sample of  
739 **messageOrderSemantics** follows.  
740

741 `<QualityOfServiceInfo deliverySemantics="OnceAndOnlyOnce"`  
742 `messageOrderSemantics="Guaranteed"/>`

743 **8.4.7.2 DeliveryReceiptRequested attribute**

744 The **deliveryReceiptRequested** element attribute, if present, over-rides the value of the same  
745 parameter in the CPA. If not present then the value in the CPA MUST be used.

746 The **deliveryReceiptRequested** parameter/element MUST be used by a *From Party* MSH to  
747 indicate whether a message received by the *To Party* MSH should result in the *To Party* MSH  
748 returning an acknowledgment message containing an **Acknowledgment** element with a **type** of  
749 **deliveryReceipt**.

750 The **deliveryReceiptRequested** parameter/element is frequently used to help implement  
751 Reliable Messaging (see section 10) although it can be used independently.

752 Before setting the value of **deliveryReceiptRequested**, the *From Party* SHOULD check the  
753 **deliveryReceiptSupported** parameter for the *To Party* in the CPA to make sure that its value is  
754 compatible.

755 Valid values for **deliveryReceiptRequested** are:

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

759 When a *To Party* MSH receives a message with **deliveryReceiptRequested** not set to **None**  
760 then it should check if it is able to support the type of Delivery Receipt requested.

761 If the *To Party* MSH can produce the Delivery Receipt of the type requested, then it MUST return  
 762 to the *From Party* on the message just received, a message containing an  
 763 **AcknowledgementAcknowledgment** element with the value of the **type** attribute set to  
 764 **DeliveryReceipt**.

765 If the *To Party* cannot return a Delivery Receipt of the type requested then it MUST report the  
 766 error to the *From Party* using an **ErrorCode** of **NotSupported** and a **Severity** of **Error**.

#### 767 8.4.7.3 syncReplyMode attribute

768 The **syncReplyMode** is an optional attribute that indicates whether a response to a message  
 769 must be returned at the same time as any **acknowledgementacknowledgments**. It has two values:

- 770 • **True** which indicates that the MSH that receives the message MUST get the message  
 771 processed by the application or other process that needs to process it before the MSH  
 772 sends any response to the original message, or
- 773 • **False** which indicates that an **acknowledgementacknowledgment** to the message MAY  
 774 be sent separately before processing of the message by the application or other process.

775 The default value is **False**.

#### 776 8.4.7.4 TimeToLive attribute

777 The **TimeToLive** is an optional attribute in the header that conforms to [ISO8601] and indicates  
 778 the time by which a message should be delivered to the *To Party* Message Service Handler.

779 When setting a value for **TimeToLive** it is RECOMMENDED that the *From Party* takes into  
 780 account the accuracy of its own internal clocks as well as the **mshTimeAccuracy** parameter for  
 781 the Receiver MSH (see section 10.6.5.3) that indicates the accuracy to which a MSH will keep its  
 782 internal clocks.

783 How a MSH ensures that its internal clocks are kept sufficiently accurate is an implementation  
 784 decision.

785 If a MSH receives a Message where **TimeToLive** has expired the MSH MUST:

- 786 • send a *Message* to the *From Party* MSH, reporting that the **TimeToLive** of the message  
 787 has passed
- 788 • NOT forward the message to another MSH or application/other system that should  
 789 receive the message.

790 The *message* reporting the error MUST contain an **ErrorCode** set to **TimeToLiveExpired**, and a  
 791 **severity** attribute set to **Error**

792 In this context the **TimeToLive** has expired if the time of the internal clock of the MSH that  
 793 receives a message is greater than the value of **TimeToLive** for the *Message*.

794 If **TimeToLive** is not present then it MUST be assumed that **TimeToLive** is infinite and therefore  
 795 checks for message expiry are unnecessary.

#### 796 8.4.8 SequenceNumber element

797 The **SequenceNumber** is an element that indicates the sequence in which messages must be  
 798 processed by a *To Party* receiving MSH. The **SequenceNumber** is unique within the  
 799 **ConversationId** and *From Party* MSH. It is set to zero on the first message from that MSH for a  
 800 **Conversation** and then incremented by one for each subsequent message sent. The  
 801 **SequenceNumber** element MUST appear only when **deliverySemantics** is **OnceAndOnlyOnce**  
 802 and **messageOrderSemantics** is **Guaranteed**. If it does not, then there is an error that must be  
 803 reported to the *From Party* MSH with an **errorCode** of **Inconsistent** and a **severity** of **Error**.

804 A *To Party* MSH that receives a message with a **SequenceNumber** set MUST NOT pass the  
 805 message to an application as long as the storage required to save out-of-sequence messages is

806 within the implementation defined limits and until all the messages with lower  
 807 **SequenceNumbers** have been received and passed to the application.

808 If the implementation defined limit for saved out-of-sequence messages is reached, then the *To*  
 809 *Party* MSH MUST indicate a delivery failure to the *From Party* MSH with **errorCode** set to  
 810 **DeliveryFailure** and **severity** set to **Error** (see section [Error! Reference source not found.44](#)).

811 The **SequenceNumber** element is an integer value that is incremented (e.g. 0, 1, 2, 3, 4...) for  
 812 each *From Party* application-prepared message sent to the *To Party* application in the  
 813 **ConversationId**. The next value of 99999999 in the increment is "0". The Sequence Number  
 814 consists of ASCII numerals in the range 0-99999999. In following cases, the Sequence Number  
 815 takes the value "0":

- 816 1) *First message from the within the Conversation*
- 817 2) *First message after resetting Sequence Number information by the from Party MSH*
- 818 3) *First message after wraparound (next value after 99999999)*

819 The **SequenceNumber** element has a single attribute, **Status**. This attribute is an enumeration, which  
 820 SHALL have one of the following values:

- 821 • **Reset** – the Sequence Number is reset as shown in 1 or 2 above
- 822 • **Continue** – the Sequence Number continues sequentially (including 3 above)

823 When the Sequence Number is set to "0" because of 1 or 2 above, the **Status** attribute of the  
 824 messages MUST be set to "Reset". In all other cases, including 3 above, the **Status** attribute  
 825 MUST be set to "Continue". Before the *From Party* resets the SequenceNumber of a  
 826 *Conversation*, the Sender MUST wait for receiving of all the *Acknowledgement Messages* for  
 827 *Messages* previously sent for the *Conversation*. Only when all the sent Messages are  
 828 acknowledged, can the *From Party* reset the **SequenceNumber**. An example of a Sequence  
 829 Number follows.

```
830 <SequenceNumber Status="Reset">0</SequenceNumber>
```

832 **8.4.88.4.9 Description element**

833 The **Description** element ~~MAY be is~~ present zero or more times as a child element of the  
 834 Header. Its purpose is to provide a human readable description of the purpose or intent of the  
 835 message. The language of the description is defined by a required **xml:lang** attribute. The  
 836 **xml:lang** attribute MUST comply with the rules for identifying languages specified in [XML]. Each  
 837 occurrence SHOULD have a different value for **xml:lang**. ~~This element is provided solely for the~~  
 838 ~~purposes of providing a human readable description of the purpose or intent of the message.~~

839 **8.4.98.4.10 #wildcard element**

840 In support of allowing an ebXML Message to be extended to include element content from a  
 841 foreign namespace, a **#wildcard** element has been provided. Additional element content MAY be  
 842 added to the **Header** element immediately following the **MessageData** element. Such additional  
 843 element content MUST be namespace-qualified in accordance with [XMLNamespaces].

844 [Refer to section 8.2.3.9 for discussion of #wildcard element handling.](#)

846 **8.4.108.4.11 Header sample**

847 The following fragment demonstrates the structure of the **Header** element of the **ebXMLHeader**  
 848 document:

```
849 <Header id="N01">
850 <From>
851 <PartyId type="uri">...</PartyId>
852 </From>
853 <To>
```

```

855 <PartyId type="userType">...</PartyId>
856 </To>
857 <CPAId>http://www.ebxml.org/cpa/123456</CPAId>
858 <ConversationId>987654321</ConversationId>
859 <Service type="myservicetypes">QuoteToCollect</Service>
860 <Action>NewPurchaseOrder</Action>
861 <MessageData>
862   <MessageId>UUID-2</MessageId>
863   <Timestamp>20000725T121905.000Z</Timestamp>
864   <RefToMessageId>UUID-1</RefToMessageId>
865 </MessageData>
866 <ReliableMessagingInfoQualityOfServiceInfo deliverySemantics="BestEffort"/>
867 </Header>

```

## 868 8.5 RoutingHeaderList element

869 A **RoutingHeaderList** [element](#) consists of one or more **RoutingHeader** elements. Exactly one  
870 **RoutingHeader** is appended to the **RoutingHeaderList**, following any pre-existing  
871 **RoutingHeader** before transmission of a message over a data communication protocol.

872 The **RoutingHeaderList** element MAY be omitted from the header if:

- 873 • the message is being sent over a single hop (see section 8.5.2), and
- 874 • the message is not being sent reliably (see section 10)

### 875 8.5.1 Routing Header Element

876 The **RoutingHeader** element contains information about a single transmission of a message  
877 between two Parties. If a message traverses multiple hops by passing through some type of  
878 intermediate system between the *From Party* and the *To Party*, then each transmission over each  
879 hop results in the addition of a new Routing Header element.

880 The **RoutingHeader** element is a composite element comprised of the following subordinate  
881 elements:

- 882 • **SenderURI**
- 883 • **ReceiverURI**
- 884 • **ErrorURI**
- 885 • **Timestamp**
- 886 • **SequenceNumber**
- 887 • **#wildcard**

888 [The RoutingHeader element MAY contain either or both of the following attributes:](#)

- 889 • [reliableMessagingMethod](#)
- 890 • [intermediateAckRequested](#)

891 [\[These attributes are added to support the prototype Reliable Messaging content in this  
892 specification—see note at beginning of section 10\]](#)

#### 893 8.5.1.1 SenderURI element

894 This element contains the URI of the message's Sender Messaging Service Handler. The  
895 recipient of the message, unless there is another URI more specifically identified within the CPA,  
896 uses the URI to send a message, when required that:

- 897 • responds to an earlier message
- 898 • acknowledges an earlier message
- 899 • reports an error in an earlier message.

### 900 **8.5.1.2 ReceiverURI element**

901 This element contains the URI of the Receiver's Messaging Service Handler URI. It is the URI to  
902 which the Sender sends the message.

### 903 **8.5.1.3 ErrorURI element**

904 This URI, if present, identifies the URI that is used for reporting errors. If it is not present then  
905 errors are reported by sending a message to the **SenderURI**.

### 906 **8.5.1.4 Timestamp element**

907 The **Timestamp** element is the time the individual **RoutingHeader** was created. It is in the same  
908 format as in the **Timestamp** element in the **MessageData** element.

### 909 **8.5.1.5 SequenceNumber element**

910 The **SequenceNumber** is an optional element that indicates the sequence in which messages  
911 must be processed by a receiving MSH. The SequenceNumber is unique within the  
912 **ConversationId** and Sender MSH. It is set to one on the first message from that MSH for a  
913 Conversation and then incremented by one for each subsequent message sent.

914 Preservation of message sequence MUST be used with **deliverySemantics** of  
915 **OnceAndOnlyOnce** otherwise there is an error.

916 A MSH that receives a message with a **SequenceNumber** set MUST NOT pass the message to  
917 an application as long as the storage required to save out-of-sequence messages is within the  
918 implementation defined limits and until all the messages with lower **SequenceNumbers** have  
919 been received and passed to the application.

920 If the implementation defined limit for saved out-of-sequence messages is reached, then the  
921 Receiving MSH MUST indicate a delivery failure to the Sending MSH with **errorCode** set to  
922 **DeliveryFailure** and **severity** set to **Error** (see section 10.5).

### 923 **8.5.1.6 #wildcard element**

924 Refer to section 8.2.3.9 for discussion of #wildcard element handling.

### 925 **8.5.1.7 reliableMessagingMethod attribute**

926 The **reliableMessagingMethod** attribute is an enumeration that SHALL have one of the following  
927 values:

- 928 • ebXML
- 929 • Transport

930 The default implied value for this attribute is "ebXML". Refer to section 10.1.2 for discussion of  
931 the use of this attribute.

### 932 **8.5.1.8 intermediateAckRequested attribute**

933 The **intermediateAckRequested** attribute is an enumeration that SHALL have one of the  
934 following values:

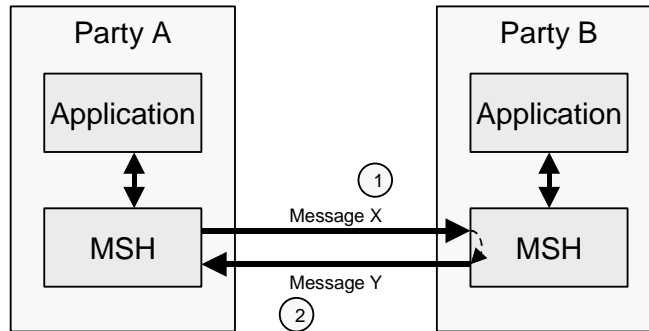
- 935 • Signed
- 936 • Unsigned
- 937 • None

938 [The default implied value for this attribute is "None". Refer to section 10.1.2 for discussion of the](#)  
 939 [use of this attribute.](#)

940 [This MAY contain any namespace-qualified element content belonging to a foreign namespace.](#)

941 **8.5.2 Single Hop Routing Header Sample**

942 A single hop message and its return is illustrated by the diagram below.



943

944 **Figure 8-1 Single Hop Message**

945 The content of the corresponding messages could include:

- Transmission 1 - Message X From Party A To Party B

946

```

947 <Header id="...">
948   <From>urn:myscheme.com:id:PartyA-id</From>
949   <To>urn:myscheme.com:id:PartyB-id</To>
950   <ConversationId>219cdj89dj2398djfjn</ConversationId>
951   ...
952   <MessageData>
953     <MessageId>29dmridj103kvna</MessageId>
954     ...
955   </MessageData>
956   ...
957 </Header>
958 <RoutingHeaderList id="...">
959   <RoutingHeader>
960     <SenderURI>url:PartyA.com/PartyAMsh</SenderURI>
961     <ReceiverURI>url:PartyB.com/PartyBmsh</ReceiverURI>
962     <Timestamp>20001216T21:19:35.145Z-8</Timestamp>
963   </RoutingHeader>
964 </RoutingHeaderList>
    
```

965

- Transmission 2 - Message Y From Party B To Party A

966

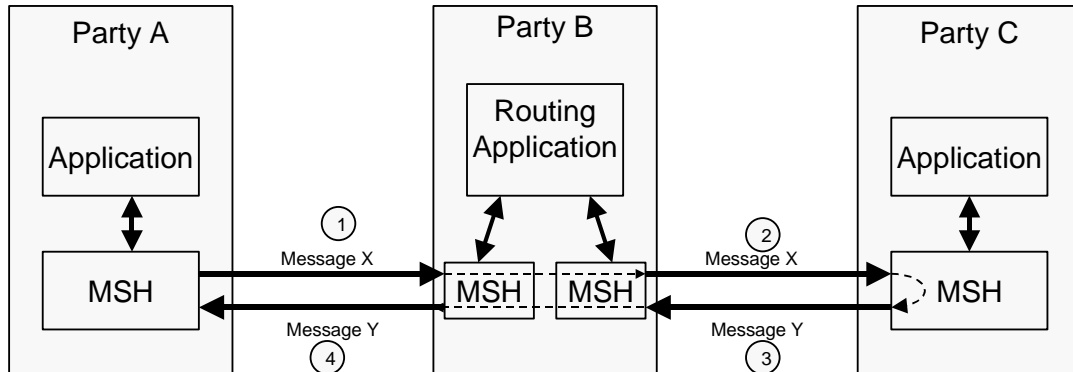
```

967 <Header id="...">
968   <From>urn:myscheme.com:id:PartyB-id</From>
969   <To>urn:myscheme.com:id:PartyA-id</To>
970   <ConversationId>219cdj89dj2398djfjn</ConversationId>
971   ...
972   <MessageData>
973     <MessageId>eis99dk4mvzlgghasi</MessageId>
974     <RefToMessageId>29dmridj103kvna</RefToMessageId>
975     ...
976   </MessageData>
977   ...
978 </Header>
979 <RoutingHeaderList id="...">
980   <RoutingHeader>
981     <SenderURI>url:PartyA.com/PartyAMsh</SenderURI>
982     <ReceiverURI>url:PartyB.com/PartyBmsh</ReceiverURI>
983     <Timestamp>20001216T21:20:05.274Z-6</Timestamp>
984   </RoutingHeader>
    </RoutingHeaderList>
    
```

984

985 **8.5.3 Multi-hop Routing Header Sample**

986 Multi-hop messages are not sent directly from one party to another, instead they are sent via an  
 987 intermediate party. This is illustrated by the diagram below.



988

989 **Figure 8-2 Multi-hop Message**

990 The content of the corresponding messages could include:

- Transmission 1 - Message X From Party A To Party B

991  
992  
993  
994  
995  
996  
997  
998  
999  
1000  
1001  
1002  
1003  
1004  
1005  
1006  
1007  
1008  
1009

```
<Header id="...">
  <From>urn:myscheme.com:id:PartyA-id</From>
  <To>urn:myscheme.com:id:PartyC-id</From>
  <ConversationId>219cdj89dj2398djfjn</ConversationId>
  ...
  <MessageData>
    <MessageId>29dmridj103kvna</MessageId>
    ...
  </MessageData>
  ...
</Header>
<RoutingHeaderList id="...">
  <RoutingHeader>
    <SenderURI>url:PartyA.com/PartyAMsh</SenderURI>
    <ReceiverURI>url:PartyB.com/PartyBMsh</ReceiverURI>
    <Timestamp>20001216T21:19:35.145Z-8</Timestamp>
  </RoutingHeader>
</RoutingHeaderList>
```

- Transmission 2 - Message X From Party B To Party C

1010  
1011  
1012  
1013  
1014  
1015  
1016  
1017  
1018  
1019  
1020  
1021  
1022  
1023  
1024  
1025  
1026  
1027  
1028  
1029  
1030  
1031  
1032  
1033

```
<Header id="...">
  <From>urn:myscheme.com:id:PartyA-id</From>
  <To>urn:myscheme.com:id:PartyC-id</From>
  <ConversationId>219cdj89dj2398djfjn</ConversationId>
  ...
  <MessageData>
    <MessageId>29dmridj103kvna</MessageId>
    ...
  </MessageData>
  ...
</Header>
<RoutingHeaderList id="...">
  <RoutingHeader>
    <SenderURI>url:PartyA.com/PartyAMsh</SenderURI>
    <ReceiverURI>url:PartyB.com/PartyBMsh</ReceiverURI>
    <Timestamp>20001216T21:19:35.145Z-8</Timestamp>
  </RoutingHeader>
  <RoutingHeader>
    <SenderURI>url:PartyB.com/PartyAMsh</SenderURI>
    <ReceiverURI>url:PartyC.com/PartyBMsh</ReceiverURI>
    <Timestamp>20001216T21:19:45.483Z-6</Timestamp>
  </RoutingHeader>
</RoutingHeaderList>
```

1034 Message Y would be similar to Message X except that the direction of transmission is reversed.

## 1035 8.6 ApplicationHeaders Element

1036 The **ApplicationHeaders** element supports the extension of an ebXML Message through the  
1037 inclusion of additional XML elements that belong to a foreign namespace, as child elements of  
1038 the **ApplicationHeaders** element.

1039 Any additional element content **MUST** be namespace-qualified in accordance with  
1040 [XMLNamespaces].

1041 An MSH implementation **MUST** make the information content of the **ApplicationHeaders**  
1042 element available to the application or application services layer of software. How this is done is  
1043 an implementation decision but conformance to the ebXML Service Interface specification (to be  
1044 defined) is recommended.

1045 The **ApplicationHeaders** element has a single attribute called **mustUnderstand**. This attribute  
1046 has two possible values **true** and **false**. The default value for the **mustUnderstand** attribute is  
1047 false.

1048 An **ApplicationHeaders** element that has a **mustUnderstand** set to a value of **true** means that a  
1049 receiving MSH **MUST** be capable of understanding the meaning of the namespace-qualified  
1050 element content. If the content is not understood, the receiving MSH **MUST** respond with a  
1051 message that includes an **errorCode** of **NotSupported** in an **Error** element as defined in section  
1052 8.8.

### 1053 8.6.1 ApplicationHeaders sample

```
1054 <ApplicationHeaders mustUnderstand="true">
1055   <foo:ProprietaryStuff
1056     xmlns:foo="http://www.example.com/ebxml-msh-extensions">...
1057   </foo:ProprietaryStuff>
1058 </ApplicationHeaders>
```

## 1059 8.7 StatusData Element

1060 The **StatusData** element is used by one MSH to respond to a request on the status of the  
1061 processing of a message that was previously sent (see also section 9.1).

1062 The **StatusData** element consists of the following elements and attributes:

- 1063 • a **RefToMessageld** element that contains the **Messageld** of the message whose status  
1064 is being reported
- 1065 • a **Timestamp** element. This contains the time that the message, whose status is being  
1066 reported, was received. This **MUST** be omitted if the message whose status is being  
1067 reported is **NotRecognized** or the request was **Unauthorized**
- 1068 • a **ForwardURI** element. This **MUST** only be present if **messageStatus** is set to  
1069 **Forwarded**. If present it indicates the URI of the **ReceiverURI** to which the message was  
1070 forwarded
- 1071 • a **messageStatus** attribute that is set to one of the following values:
  - 1072 - **Unauthorized** – the Message Status Request is not authorized or accepted
  - 1073 - **NotRecognized** – the message identified by the **RefToMessageld** element in the  
1074 **StatusData** element is not recognized
  - 1075 - **Received** – the message identified by the **RefToMessageld** element in the  
1076 **StatusData** element has been received by the MSH, but has not been processed by  
1077 an application or forwarded to another MSH
  - 1078 - **Processed** – the message identified by the **RefToMessageld** element in the  
1079 **StatusData** element has been received by the MSH for the To Party on the original  
1080 message, and has been passed to the application or other process that is to handle it

1081 - **Forwarded** – the message identified by the **RefToMessageId** element in the  
1082 **StatusData** element has been received by the MSH, and has been forwarded to  
1083 another MSH

## 1084 8.8 ErrorList Element

1085 The existence of an **ErrorList** element indicates that the message that is identified by the  
1086 **RefToMessageId** in the header has an error.

1087 The **ErrorList** element consists of one or more **Error** elements and the following two attributes:

- 1088 • **id** attribute
- 1089 • **highestSeverity** attribute

1090 If there are no errors to be reported then the **ErrorList** element MUST NOT be present.

### 1091 8.8.1 id attribute

1092 The **id** attribute uniquely identifies the **ErrorList** element within the document.

### 1093 8.8.2 highestSeverity attribute

1094 The **highestSeverity** attribute contains the highest severity of any of the **Error** elements.  
1095 Specifically, if any of the **Error** elements has a **severity** of **Error** then **highestSeverity** must be  
1096 set to **Error** otherwise set **highestSeverity** to **Warning**.

### 1097 8.8.3 Error element

1098 An **Error** element consists of the following [attributes](#):

- 1099 • **codeContext** [attribute](#)
- 1100 • **errorCode** [attribute](#)
- 1101 • **severity** [attribute](#)
- 1102 • **location** [attribute](#)
- 1103 • **xml:lang** [attribute](#)
- 1104 • **errorMessage** [attribute](#)
- 1105 • **softwareDetails** [attribute](#)

#### 1106 8.8.3.1 codeContext attribute

1107 The [REQUIRED](#) **codeContext** attribute identifies the namespace or scheme for the **errorCodes**.  
1108 It MUST be a URI. Its default value is **http://www.ebxml.org/messageServiceErrors**. If it is  
1109 does not have the default value then it indicates that an implementation of this specification has  
1110 used its own **errorCodes**.

1111 Use of non ebXML values for **errorCodes** is NOT RECOMMENDED. In addition, an  
1112 implementation of this specification MUST NOT use its own **errorCodes** if an existing **errorCode**  
1113 as defined in section 8.8.5 has the same or very similar meaning.

#### 1114 8.8.3.2 errorCode attribute

1115 The required **errorCode** attribute indicates the nature of the error in the *message in error*. Valid  
1116 values for the **errorCode** and a description of the code's meaning are given in section 8.8.5.

#### 1117 8.8.3.3 severity attribute

1118 The required **severity** attribute indicates the severity of the error. Valid values are:

- 1119 • **Warning** - This indicates that although there is an error, other messages in the  
1120 conversation will still be generated in the normal way.

- 1121       • **Error** - This indicates that there is an unrecoverable error in the message and no further  
 1122       messages will be generated as part of the conversation.

1123       **8.8.3.4 location attribute**

1124       The **location** attribute points to the part of the message that is in error.  
 1125       If an error exists in the ebXML Header document and the document is “well formed” (see [XML]),  
 1126       then the content of the **location** attribute MUST be an [XPointer].  
 1127       If the ebXML Header document is not “well formed” then the location attribute MUST be omitted.  
 1128       If the error is associated with the MIME envelope that wraps the ebXML Header Document and  
 1129       the ebXML Payload, then **location** id contains the content-id of the MIME part that is in error, in  
 1130       the format cid:23912480wsr, where the text after the “:” is the value of the MIME part’s content-  
 1131       id.  
 1132       The **location** attribute MUST NOT be used to point to errors inside the ebXML Payload Container  
 1133       as the method of reporting errors in the ebXML Payload Container is application dependent.

1134       **8.8.3.5 errorMessage attribute**

1135       The **errorMessage** attribute provides a narrative description of the error in the language defined  
 1136       by the **xml:lang** attribute. Typically, it will be the message generated by the XML parser or other  
 1137       software that is validating the message. This means that the value of the attribute is defined by  
 1138       the vendor/developer of the software, that generated the Error element.  
 1139       The **xml:lang** must comply with the rules for identifying languages specified in [XML].  
 1140       The **errorMessage** attribute MAY be omitted.  
 1141       <DB>Do we want to allow multiple errorMessage elements in different languages, e.g. so that if  
 1142       you send a message to Switzerland you could send it in French, German and Italian?</DB>

1143       **8.8.3.6 softwareDetails attribute**

1144       The **softwareDetails** attribute contains a value that is set by the vendor/developer of the software  
 1145       that generated the **Error** element. It SHOULD contain data that enables the vendor/developer as  
 1146       well as the recipient of the message to identify the precise location in their software and the set of  
 1147       circumstances that caused the software to generate a *message reporting the error*. It is  
 1148       RECOMMENDED that this element include plain text separated by punctuation to identify:  
 1149       • the name of the software vendor;  
 1150       • the name, version and release number of the software that generated the ebXML Error  
 1151       Document  
 1152       • the part of the software that caused the error to be generated that can be used by the  
 1153       Software Vendor to identify the circumstances that caused the error

1154       If any part of the **softwareDetails** attribute contains text that is readable by a human, then it  
 1155       SHOULD be in the language identified by **xml:lang**.

1156       **8.8.4 Examples**

1157       An example of an **ErrorList** element is given below.

```

1158 <ErrorList id='3490sdo9', highestSeverity="error">
1159   <Error errorCode='UnableToParse', severity="Error", location=cid:21398adhiwqe, xml:lang="us-
1160   en", errorMessage='XSD parser error - document not parsable', softwareDetails='Software
1161   Development Corp.; ebXML Connector!!! v2.7, build 2.7313; Ref HA' />
1162   <Error ... />
1163 </ErrorList>
1164 
```

1165 **8.8.5 errorCode values**

1166 This section describes the **ErrorCodes** (see section 8.8.3.2) that are used in a *message*  
 1167 *reporting an error*. They are described in a table with three headings:

- 1168 • the first column contains the value to be used as an **errorCode**, e.g. **UnableToParse**
- 1169 • the second column contains a "Short Description" of the **errorCode**. Note that this  
 1170 narrative MUST NOT be used in the **errorMessage** attribute.
- 1171 • the third columns contains a "Long Description" that provides an explanation of the  
 1172 meaning of the error and provides guidance on when the particular **ErrorCode** should be  
 1173 used.

1174 It is RECOMMENDED that implementers of software that conforms to this specification make  
 1175 available to a user that is being informed of the error: the value of the **errorCode**, the "Short  
 1176 Description" and optionally the "Long Description".

1177 It is also RECOMMENDED that the "Short Description" and the "Long Description" are translated  
 1178 into the preferred language of the user if this is known.

1179 **8.8.6 Reporting Errors in the ebXML Header Document**

1180 The following list contains error codes that can be associated with the *ebXML Header Document*:  
 1181

Error Code	Short Description	Long Description
<b>UnableToParse</b>	XML not well formed or invalid.	The XML document is not well formed or not valid and cannot be successfully parsed. See [XML] for the meaning of "well formed" and "not valid".
<b>ValueNotRecognized</b>	Element content or attribute value not recognized.	Although the document is well formed and valid, the element/attribute contains a value that could not be recognized and therefore could not be used by the ebXML Message Service
<b>NotSupported</b>	Element or attribute not supported	Although the document is well formed and valid, an element or attribute is present that: <ul style="list-style-type: none"> <li>• is consistent with the rules and constraints contained in this specification, but</li> <li>• is not supported by the ebXML Message Service that is processing the message.</li> </ul>
<b>Inconsistent</b>	Element content or attribute value inconsistent with other elements or attributes.	Although the document is well formed and valid, according to the rules and constraints contained in this specification the content of an element or attribute is inconsistent with the content of other elements or their attributes.
<b>OtherXml</b>	Other error in an element content or attribute value.	Although the document is well formed and valid, the element content or attribute value contains values that do not conform to the rules and constraints contained in this specification and is not covered by other error codes. The errorMessage attribute should be used to indicate the nature of the problem.

1182 **8.8.7 Non-XML Document Errors**

1183 The following are error codes that identify errors that are not associated with the ebXML Header  
 1184 Document:

1185

Error Code	Short Description	Long Description
<b>MessageTooLarge</b>	Message too large	The message is too large to be processed by the ebXML Message Service.
<b>MimeProblem</b>	A MIME error has occurred	An error has been detected in the structure or format of a MIME part of the message. For example: <ul style="list-style-type: none"> <li>• Missing MIME Part. Although the MIME message is correctly structured, a MIME part is missing that should have been present if the rules and constraints contained in this specification are followed</li> <li>• Unexpected MIME Part. Unexpected MIME part. Although the MIME message is correctly structured, a MIME part is present that is not expected in the particular context according to the rules and constraints contained in this specification</li> </ul>
<b>DeliveryFailure</b>	Message Delivery Failure	A message has been received that either probably or definitely could not be sent to its next destination. Note that if <b>severity</b> is set to <b>Warning</b> then there is a small probability that the message was delivered.
<b>TimeToLiveExpired</b>	Message Time To Live Expired	A message has been received that arrived after the time specified in the <b>TimeToLive</b> element of the <b>Header</b> element
<b>SecurityFailure</b>	Message Security Checks Failed	Validation of signatures or checks on the authenticity or authority of the sender of the message have failed.
<b>Unknown</b>	Unknown Error	Indicates that an error has occurred that is not covered explicitly by any of the other errors. The <b>errorMessage</b> attribute should be used to indicate the nature of the problem.

1186 **8.9 Acknowledgment Element**

1187 The Acknowledgment element is an optional element that is used by one Message Service  
 1188 Handler to indicate that another Message Service Handler has received a message.

1189 For clarity two terms are defined:

- 1190 • *message being acknowledged*. This is the Message that is has been received by a MSH  
 1191 that is now being acknowledged
- 1192 • *acknowledgment message*. This is the message that acknowledges that the *message*  
 1193 *being acknowledged* has been received.

1194 The *message being acknowledged* is identified by the **RefToMessageId** contained in the  
 1195 **MessageData** element contained within the **Header** Element of the acknowledgment message  
 1196 containing the value of the **MessageId** of the message being acknowledged.

1197 The **Acknowledgment** element consists of the following:

- 1198 • a **Timestamp** element
- 1199 • a **From** element
- 1200 • a **type** attribute
- 1201 • a **signed** attribute

### 1202 **8.9.1 Timestamp element**

1203 The **Timestamp** element is a value representing the time that the *message being acknowledged*  
1204 was received by the Party generating the *acknowledgment message*. It must conform to [ISO-  
1205 8601]. <DB>Do we make this conform to XML Schema *timeInstant*</DB>

### 1206 **8.9.2 From element**

1207 This is the same element as the **From** element within **Header** element (see section 8.4.1).  
1208 However, when used in the context of an Acknowledgment Element, it contains the identifier of  
1209 the *Party* that is generating the *acknowledgment message*.

1210 If the **From** element is omitted then the *Party* that is sending the element is identified by the **From**  
1211 element in the **Header** element.

### 1212 **8.9.3 type attribute**

1213 The **type** attribute indicates who sent the *acknowledgment message*. It MUST contain either:  
1214 • **DeliveryReceipt** - indicates that the *acknowledgment message* was generated by the *To*  
1215 *Party* identified by the **To** element of the *message being acknowledged*, or  
1216 • **IntermediateAck** - indicates that the *acknowledgment message* was generated by a  
1217 *Party* that is not the *To Party* identified by the **To** element of the *message being*  
1218 *acknowledged*. Typically this will be a *Party* that has received the message and is  
1219 forwarding it to either the *To Party* or another *Party* with the intention that the message is  
1220 sent to the *To Party*.

1221 The default value for **type** is **DeliveryReceipt**.

### 1222 **8.9.4 signed attribute**

1223 The **signed** attribute indicates whether the *acknowledgment message* is digitally signed. It MUST  
1224 contain either:

- 1225 • **True** - indicates that the *acknowledgment message* is digitally signed, or
- 1226 • **False** - indicates that the *acknowledgment message* is not digitally signed

1227 The default value for **signed** is **False**.

1228 See section [12](#) for details on what should be signed and how a signature that signs an  
1229 *acknowledgment message* should be checked.

## 1230 **8.10 Signature Element**

1231 TBD

## 1232 9 Message Service Handler Services

1233 [The Message Service Handler Services section has not been agreed to by the  
1234 membership of the TRP Project Team; however, it is being included to provide a basis for  
1235 POC developers of MSH implementations. Implementers MUST be prepared for some  
1236 change to the content of this section.]

1237 The Message Service Handler MUST support two services that are designed to help provide  
1238 smooth operation of a Message Handling Service implementation:

- 1239 • Message Status Request
- 1240 • Message Service Handler Ping

1241 Each service is described below:

### 1242 9.1 Message Status Request Service

1243 The Message Status Request Service consists of the following:

- 1244 • sending a Message Status Request message to a Message Service Handler (MSH)  
1245 about a message previously sent
- 1246 • the Message Service Handler that receives the request sending a Message Status  
1247 Response message in return.

#### 1248 9.1.1 Message Status Request Message

1249 A Message Status Request message consists of no *ebXML Payload* and the following elements  
1250 in the ebXML Header:

- 1251 • A **Header** element
- 1252 • A **RoutingHeaderList** element
- 1253 • A **Signature** element

1254 The **RoutingHeaderList** and the **Signature** elements MAY be omitted (see sections 8.5 and  
1255 8.10).

1256 The **Header** element MUST contain the following:

- 1257 • a **From** element that identifies the party that created the message status request  
1258 message
- 1259 • a **To** element that identifies a Party that should receive the message. If a **RoutingHeader**  
1260 was present on the message whose status is being checked then this MUST be the  
1261 **ReceiverURI** from that message.
- 1262 • a **Service** element that contains:  
1263 **http://www.ebxml.org/namespaces/messageService/MessageStatus**
- 1264 • an **Action** element that contains **Request**

1265 The message is then sent to the *To Party*.

#### 1266 9.1.2 Message Status Response Message

1267 Once the To Party on the Message Status Request message receives the message, they MAY  
1268 generate a Message Status Response message that consists of no ebXML Payload and the  
1269 following elements in the ebXML Header.

- 1270 • a **Header** element
- 1271 • a **RoutingHeaderList** element
- 1272 • an **AcknowledgementAcknowledgment** element
- 1273 • a **StatusData** element
- 1274 • a **Signature** element

1275 The **RoutingHeaderList**, ~~**Acknowledgement**~~**Acknowledgment** and **Signature** elements MAY  
1276 be omitted (see sections 8.5, 8.9 and 8.10).

1277 The **Header** element MUST contain the following:

- 1278 • a **From** element that identifies the creator of the Message Status Response message
- 1279 • a **To** element that is set to the value of the **From** element in the Message Status Request  
1280 message
- 1281 • a **Service** element that contains:  
1282 ***http://www.ebxml.org/namespaces/messageService/MessageStatus***
- 1283 • an **Action** element that contains **Response**
- 1284 • a **RefToMessageId** that identifies the Message Status Request message.

1285 The message is then sent to the *To Party*.

### 1286 9.1.3 Security Considerations

1287 Party's that receive a Message Status Request message SHOULD always respond to the  
1288 message. However they MAY ignore the message instead of responding with **messageStatus**  
1289 set to **Unauthorized** if they consider that the sender of the message received is unauthorized.  
1290 The decision process that results in this course of action is implementation dependent.

1291 *<DB> Do we want to allow the Message Status Response to include the original response to the  
1292 message in the Payload?</DB><CF> quite possibly.</CF>*

## 1293 9.2 Message Service Handler Ping Service

1294 The Message Service Handler Ping Service enables one Message Service Handler to determine  
1295 if another MSH is operating. It consists of:

- 1296 • sending a Message Service Handler Ping message to a MSH, and
- 1297 • the MSH that receives the Ping responding with a Message Service Handler Pong  
1298 message.

### 1299 9.2.1 Message Service Handler Ping Message

1300 A Message Service Handler Ping (MSH Ping) message consists of no ebXML Payload and the  
1301 following elements in the ebXML Header:

- 1302 • A **Header** element
- 1303 • A **RoutingHeaderList** element
- 1304 • A **Signature** element

1305 The **RoutingHeaderList** and the **Signature** elements MAY be omitted (see sections 8.5 and  
1306 8.10).

1307 The **Header** element MUST contain the following:

- 1308 • a **From** element that identifies the creator of the MSH Ping message
- 1309 • a **To** element that identifies the operator of the MSH that is being sent the MSH Ping  
1310 message
- 1311 • a **Service** element that contains:  
1312 ***http://www.ebxml.org/namespaces/messageService/MSHStatus***
- 1313 • an **Action** element that contains **Ping**

1314 The message is then sent to the To Party.

### 1315 9.2.2 Message Service Handler Pong Message

1316 Once the To Party on the MSH Ping message receives the message, they MAY generate a  
1317 Message Service Handler Pong (MSH Pong) message that consists of no ebXML Payload and  
1318 the following elements in the ebXML Header.

- 1319 • a **Header** element
- 1320 • a **RoutingHeaderList** element
- 1321 • an **AcknowledgementAcknowledgment** element
- 1322 • a **Signature** element

1323 The **RoutingHeaderList**, **AcknowledgementAcknowledgment** and **Signature** elements MAY  
1324 be omitted (see sections 8.5, 8.9 and 8.10).

1325 The **Header** element MUST contain the following:

- 1326 • a **From** element that identifies the creator of the MSH Pong message
- 1327 • a **To** element that identifies a Party that generated the MSH Ping message
- 1328 • a **Service** element that contains:  
1329 ***http://www.ebxml.org/namespaces/messageService/MessageStatus***
- 1330 • an **Action** element that contains **Pong**
- 1331 • a **RefToMessageId** that identifies the MSH Ping message.

1332 The message is then sent to the To Party.

### 1333 9.2.3 Security Considerations

1334 Party's that receive a MSH Ping message SHOULD always respond to the message. However  
1335 there is a risk that some Parties might use the MSH Ping message to determine the existence of  
1336 a Message Service Handler as part of a security attack on that MSH. Therefore recipients of a  
1337 MSH Ping MAY ignore the message if they consider that the sender of the message received is  
1338 unauthorized or part of some attack. The decision process that results in this course of action is  
1339 implementation dependent.

## 1340 10 Reliable Messaging

1341 [\[The Reliable Messaging section has not been agreed to by the membership of the TRP](#)  
 1342 [Project Team; however, it is being included to provide a basis for POC developers of MSH](#)  
 1343 [implementations. Implementers MUST be prepared for some change to the content of this](#)  
 1344 [section.\]](#)

1345 Reliable Messaging defines an interoperable protocol such that the two Messaging Service  
 1346 Handlers (MSH) operated by a *From Party* and a *To Party* can “reliably” exchange messages that  
 1347 are sent using “reliable messaging” semantics.

1348 “Reliably” means that the *From Party* can be highly certain that the message sent will be  
 1349 delivered to the *To Party*. If there is a problem in sending a message then the sender resends the  
 1350 message until either the message is delivered, or the sender gives up. If the message cannot be  
 1351 delivered, for example because there has been a catastrophic failure of the *To Party*’s system,  
 1352 then the *From Party* is informed.

### 1353 10.1.1 Persistent Storage and System Failure

1354 A MSH that supports Reliable Messaging MUST keep messages that are sent or received reliably  
 1355 in *persistent storage*. In this context *persistent storage* is a method of storing data that does not  
 1356 lose information after a system failure or interruption.

1357 This specification recognizes that different degrees of resilience may be realized depending on  
 1358 the technology that is used to persist the data. However, as a minimum, persistent storage that  
 1359 has the resilience characteristics of a hard disk (or equivalent) SHOULD be used. It is strongly  
 1360 RECOMMENDED though that implementers of this specification use technology that is resilient to  
 1361 the failure of any single hardware or software component.

1362 Even after a system interruption or failure, a MSH MUST ensure that messages in persistent  
 1363 storage are processed in the same way as if the system failure or interruption had not occurred.  
 1364 How this is done is an implementation decision.

### 1365 10.1.2 Methods of Implementing Reliable Messaging

1366 ~~ebXML~~sSupport for Reliable Messaging can be implemented in one of the following two ways:

- 1367 • using the ebXML Reliable Messaging protocol, or
- 1368 • using [ebXML Header and Message structures together with](#) commercial [queuing](#)  
 1369 [transport protocol](#)-software products that are designed to provide reliable delivery of  
 1370 messages using [proprietary-alternative](#) protocols. ~~<DB>Change elsewhere</DB>~~

1371 Each of these are described below.

## 1372 10.2 ebXML Reliable Messaging Protocol

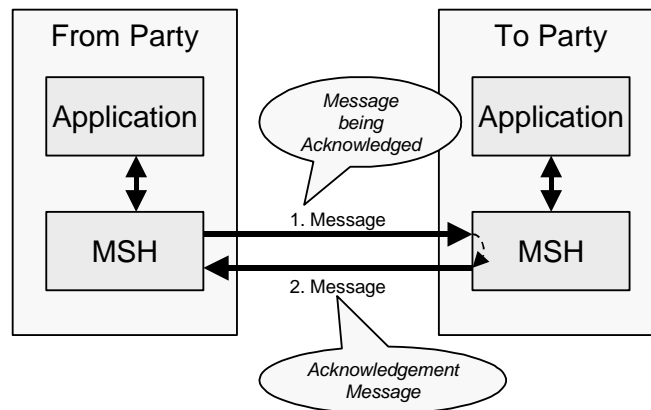
1373 ~~The ebXML Reliable Messaging Protocol is used to implement Reliable Messaging when either:~~  
 1374 ~~—no queuing transport protocol products are available, or~~  
 1375 ~~—an implementation decision has been made to use the ebXML Reliable Messaging Protocol~~  
 1376 ~~on top of a queuing transport protocol.~~

1377 ~~Use of t~~The ebXML Reliable Messaging Protocol described in this section MUST be followed if  
 1378 the *deliverySemantics* parameter/element is set to **OnceAndOnlyOnce** and the  
 1379 *ReliableMessagingMethod* parameter/element is set to **ebXML** (the default).

1380 ~~The remainder of this section describes the ebXML Reliable Messaging Protocol. In outline it~~  
 1381 ~~involves:~~

- 1382 ~~—a From Party sending a message to the To Party~~
- 1383 ~~—the To Party returning another message that references the first~~

1384 This is illustrated by the figure below.



1385

1386 **Figure 10-1 Indicating that a message has been received**

1387 The diagram above illustrates two terms that are used in the remainder of this section:

- 1388 • *message being acknowledged*. This is the Message that needs to be sent reliably and
- 1389 therefore needs to be acknowledged
- 1390 • *acknowledgment message*. This is the message that acknowledges that the message
- 1391 being acknowledged has been received.

1392 The receipt of the *acknowledgment message* indicates that the *message being acknowledged*

1393 has been sent reliably.

1394 An ~~acknowledgement~~*acknowledgment* message MUST contain a **MessageData** element with a

1395 **RefToMessageId** that contains the same value as the **MessageId** element in the *message being*

1396 *acknowledged*.

1397 ~~Note that an acknowledgment message can also contain a payload.~~

1398 A Message can be sent reliably either over:

- 1399 • a Single-hop i.e. the sending of a message directly from the *From Party's* MSH to the *To*
- 1400 *Party's* MSH without passing through any intermediate MSHs.
- 1401 • Multi-hops i.e. the sending of a message indirectly from the *From Party's* MSH to the *To*
- 1402 *Party's* MSH via one or more intermediate MSHs ~~that are not owned by or operated by on~~
- 1403 ~~behalf of either the *From Party* or the *To Party*.~~

1404 ~~Multi-hop Reliable Messaging can work either with, or without, Intermediate Acknowledgments.~~

1405 ~~See also section 8.5 on Routing Headers~~

1406 Single-hop Reliable Messaging is described first followed by Multi-hop Reliable Messaging. Note

1407 that Multi-hop Reliable Messaging is an extension of Single-hop reliable Messaging.

### 1408 10.2.1 Single-hop Reliable Messaging

1409 This section describes the REQUIRED behavior of a Message Service Handler (MSH) that is

1410 sending and/or receiving messages that support the ebXML Reliable Messaging Protocol.

#### 1411 10.2.1.1 Sending Message Behavior

1412 If a MSH is given data by an application that needs to be sent reliably then the MSH MUST do the

1413 following:

1414 ~~1)4~~ Create a message from components received from the application that includes:

- 1415 a) **deliverySemantics** set to **OnceAndOnlyOnce**, and

- 1416 b) a **RoutingHeader** element that identifies the sender and the receiver URIs
- 1417 ~~2)5)~~ Save the message in *persistent storage* (see section 10.1.1)
- 1418 ~~3)6)~~ Send the message (the *message being acknowledged*) to the *Receiver MSH*
- 1419 ~~4)7)~~ Wait for the *Receiver MSH* to return an ~~acknowledgement~~**acknowledgment** message and, if it
- 1420 does not, then resend the *identical* message as described in section 10.2.1.3
- 1421 It is RECOMMENDED that messages that are sent reliably include **deliveryReceiptRequested**
- 1422 set to **Signed** or **UnSigned**.
- 1423 If the message does not need to be sent reliably, then **deliverySemantics** MUST be set to
- 1424 **BestEffort** (the default) ~~s. Other values for the elements/attributes in the Header are as defined in~~
- 1425 ~~the CPA.~~

1426 **10.2.1.2 Receiving Message Behavior**

- 1427 If **deliverySemantics** on the received message is set to **OnceAndOnlyOnce** then do the
- 1428 following:
  - 1429 1) Check to see if the message is a duplicate (e.g.i.e. there is a message in *persistent storage*
  - 1430 that was received earlier that contains the same value for the **MessageId**)
  - 1431 2) If the message is not a duplicate then do the following:
    - 1432 a) Save the **MessageId** of the received message in *persistent storage*. As an
    - 1433 implementation decision, the whole message MAY be stored if there are other reasons
    - 1434 for doing so. ~~<DB>Need to re-look at how duplicates are detected if sequence numbers~~
    - 1435 ~~are used. </DB>~~
    - 1436 b) If the received message contains a **RefToMessageId** element then do the following:
      - 1437 i) Look for a message in *persistent storage* that has a **MessageId** that is the same as
      - 1438 the value of **RefToMessageId** on the received Message
      - 1439 ii) If a message is found in *persistent storage* then mark the persisted message as
      - 1440 delivered
      - 1441 c) If **deliveryReceiptRequested** is set to **Signed** or **UnSigned** then create an
      - 1442 **Acknowledgment** element with **type** set to **DeliveryReceipt** that identifies the *received*
      - 1443 *message*
      - 1444 d) If **syncReplyMode** is set to **True** then pass the data in the received message to the
      - 1445 application or other process that needs to process it and wait for the application to
      - 1446 produce a response.
      - 1447 e) If **deliveryReceiptRequested** is set to **Signed** or **UnSigned**, or **syncReplyMode** is set
      - 1448 to **True** then do the following:
        - 1449 i) Create a **RoutingHeader** element that identifies the sender and the receiver URIs
        - 1450 ii) Set the **RefToMessageId** to the value of the **MessageId** in the received message
        - 1451 iii) Create a *message* from the response generated by the application (if any), the
        - 1452 ~~Acknowledgement~~**Acknowledgment** element (if any) and the **RoutingHeader** that
        - 1453 includes **deliverySemantics** set to **OnceAndOnlyOnce**
        - 1454 iv) Save the message in *persistent storage* for later resending
        - 1455 v) Send the message back to the Sending MSH
        - 1456 f) If **syncReplyMode** is set to **False** then pass the data in the received message to the
        - 1457 application or other process that needs to process it. Note that, depending on the
        - 1458 application, this can result in the application generating another message to be sent (see
        - 1459 previous section).

- 1460 3) If the message is a duplicate, then do the following:
- 1461 a) Look in persistent storage for a response to the received message (i.e. it contains a
- 1462 **RefToMessageId** that matches the **MessageId** of the received message) that was *most*
- 1463 *recently sent* to the MSH that sent the received message (i.e. it has a **RoutingHeader**
- 1464 element with the greatest value of the **Timestamp**)
- 1465 b) If no message was found in *persistent storage* then ignore the received message as
- 1466 either no message was generated in response to the message, or the processing of the
- 1467 earlier message is not yet complete
- 1468 c) If a message was found in *persistent storage* then resend the persisted message back to
- 1469 the MSH that sent the received message.

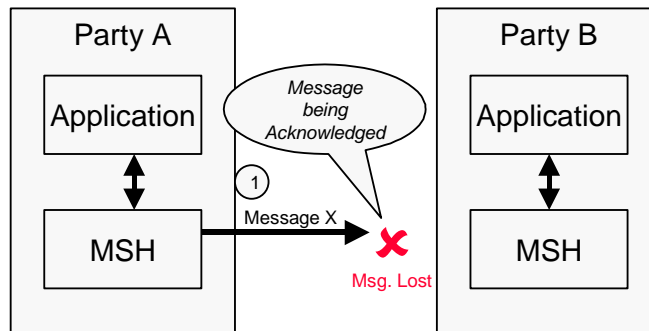
1470 **10.2.1.3 Resending Lost Messages and Duplicate Filtering**

1471 This section describes the behavior that is required by the sender and receiver of a message in

1472 order to handle ~~occurs~~ when messages are lost. A message is "lost" when a sending MSH does

1473 not receive a response to a message. For example, it is possible that a *message being*

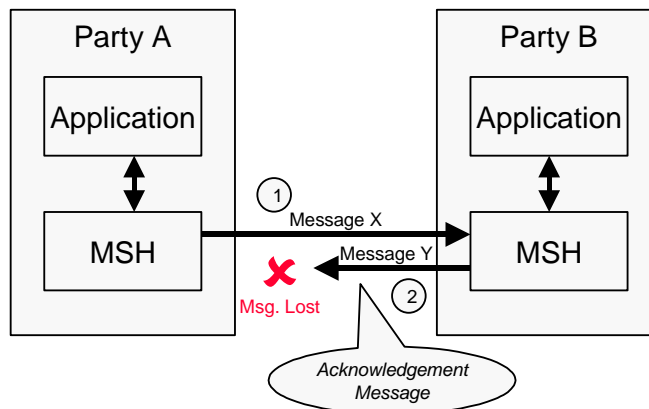
1474 *acknowledged* was lost, for example:



1475

1476 **Figure 10-2 Lost "Message Being Acknowledged"**

1477 It is also possible that the *Acknowledgment Message* was lost, for example ...

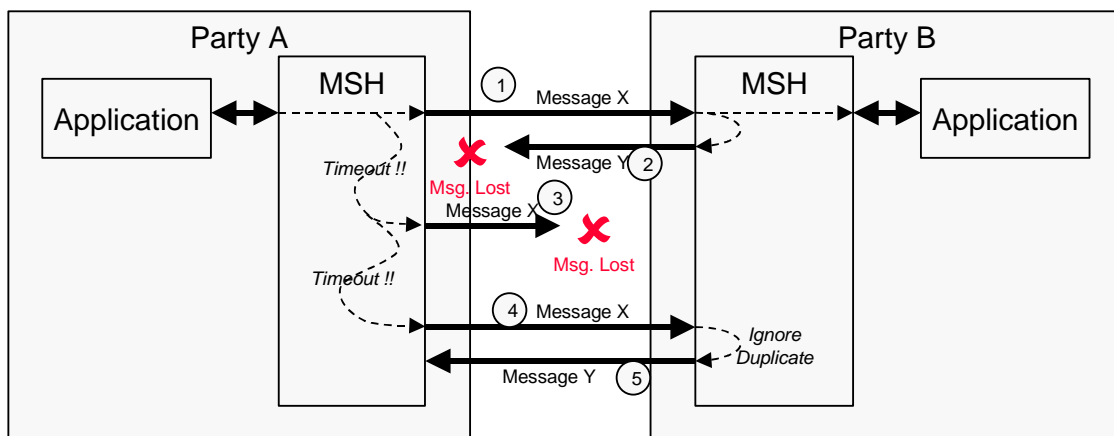


1478

1479 **Figure 10-3 Lost Acknowledgment Message**

- 1480 The rules that apply are as follows:
- 1481 1) The Sending MSH MUST resend the original message if an *Acknowledgment Message* has
- 1482 not been received from the Receiving MSH and either of the following are true:

- 1483 a) The message has not yet been resent and at least the time specified in the **timeout**  
 1484 parameter has passed since the first message was sent, or  
 1485 b) The message has been resent, and the following are both true:  
 1486 i) At least the time specified in the **retryInterval** has passed since the last time the  
 1487 message was resent, and  
 1488 ii) The message has been resent less than the number of times specified in the **retries**  
 1489 Parameter
- 1490 2) If the Sending MSH does not receive an Acknowledgment Message after the maximum  
 1491 number of retries, the Sending MSH SHOULD notify the application and/or system  
 1492 administrator function.
- 1493 ~~2) If the Sending MSH does not receive an Acknowledgment Message after the maximum number~~  
 1494 ~~of retries, the Sending MSH SHOULD notify one of the following:~~
- 1495 ~~a) The application and/or system administrator function if the Sending MSH is the From Party~~  
 1496 ~~MSH, or~~
- 1497 ~~b) The Sending MSH of the From Party, if the Sending MSH is operated by an Intermediate~~  
 1498 ~~Party (see section 10.5) <DB> This should be in multi-hop section not here. </DB>~~
- 1499 3) If the Sending MSH detects a communications protocol error that is unrecoverable at the  
 1500 transport protocol level then the Sending MSH SHOULD first attempt to resend the message  
 1501 using the same transport protocol until the number of retries has been reached, and then  
 1502 again, using a different communications protocol, if the CPA allows this. If these are not  
 1503 successful, then notify the From Party of the failure to deliver as described in section 10.5. If  
 1504 the Sending MSH detects a communications protocol error that is unrecoverable at the  
 1505 transport protocol level, the Sending MSH SHOULD first attempt to resend the message  
 1506 using a different communications protocol <DB> Should retry same protocol first </DB> if the  
 1507 CPA allows this, then if this is not successful, notify the From Party of the failure to deliver as  
 1508 described in section 10.5.



1509

1510 **Figure 10-4 Resending Lost Messages**

1511 The diagram above shows the behavior that MUST be followed by the sender of the *message*  
 1512 *being acknowledged* (e.g. Message X) and the *acknowledgment message* (e.g. Message Y).  
 1513 Specifically:

- 1514 1) The sender of the *message being acknowledged* (e.g. Party A) MUST re-send the *identical*  
 1515 *message* to the *To Party MSH* (e.g. Party B) if no *Acknowledgment Message* is received

- 1516 2) The recipient of the *message being acknowledged* (e.g. Party B), when it receives a *duplicate*  
 1517 *message*, MUST re-send to the sender of the *message being acknowledged* (e.g. Party A), a  
 1518 message identical to the *most recent message* that was sent to the recipient (i.e. Party A)
- 1519 3) The recipient of the *message being acknowledged* (e.g. Party A) MUST ignore *duplicate*  
 1520 *messages* and not forward them a second time to the application, the next MSH <DB>next  
 1521 MSH is multi-hop, should not be here. </DB> or other process that ultimately needs to receive  
 1522 them.

1523 <DB>The above also includes recipient behavior which is not part of sending behavior. Should be  
 1524 in a separate section. </DB>

1525 In this context:

- 1526 • an *identical message* is a *message* that contains, apart from perhaps an additional  
 1527 **RoutingHeader** element, the same *ebXML Header* and *ebXML Payload* as the earlier  
 1528 *message* that was sent.
- 1529 • a *duplicate message* is a *message* that contains the same **MessageId** as an earlier  
 1530 *message* that was received.
- 1531 • the *most recent message* is the *message* with the latest **Timestamp** in the **MessageData**  
 1532 element that has the same **RefToMessageId** as the duplicate *message* that has just  
 1533 been received. <DB>Chris Ferris, disagrees with resending the latest message. DB & CF  
 1534 need to go through this. </DB>

1535 Note that the Communication Protocol Envelope MAY be different. This means that the same  
 1536 *message* MAY be sent using different communication protocols and the reliable messaging  
 1537 behavior described in this section will still apply. The ability to use alternative communication  
 1538 protocols is specified in the CPA.

### 1539 10.2.2 Multi-hop Reliable Messaging

1540 Multi-hop reliable Messaging can occur either:

- 1541 • without Intermediate Acknowledgment, or
- 1542 • with Intermediate Acknowledgments

1543 One reason for using Multi-hop Reliable Messaging with Intermediate Acknowledgments is when  
 1544 the From Party that is sending a message is confident that the total time taken for ...

- 1545 • the message being acknowledged to be sent to the To Party, and
- 1546 • the acknowledgment message to be returned

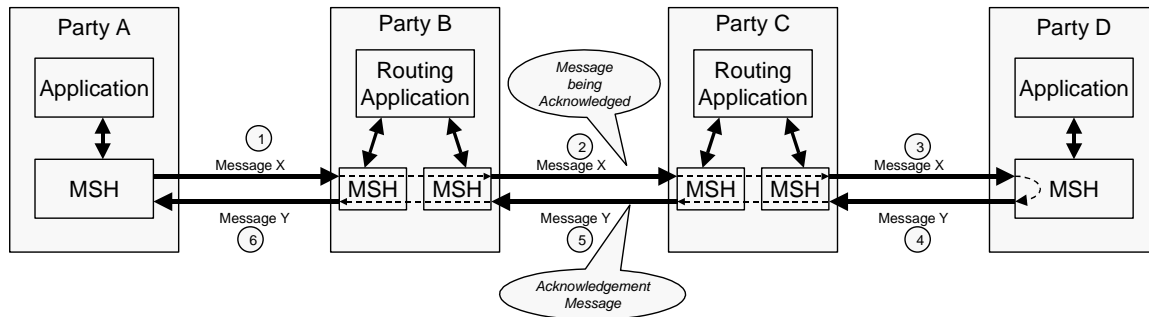
1547 ... is likely to result in the From Party resending the message being acknowledged. <DB>Chris  
 1548 thinks this is superfluous, David thinks it useful as it explains why you should do multi-hop and  
 1549 helps an implementer decide when to use it. This requires further discussion. </DB>

1550 Each of these is described below.

#### 1551 10.2.2.1 Multi-hop Reliable Messaging without Intermediate Acknowledgments

1552 Multi-hop Reliable Messaging without Intermediate Acknowledgment is identified by the  
 1553 **IntermediateAckRequested** of the *Routing Header* for the hop being set to **False** (the default).

1554 The overall message flow is illustrated by the diagram below.



1555

1556 **Figure 10-5 Multi-hop Reliable Messaging without Intermediate Acknowledgments**

1557 This is essentially the same as Single-hop Reliable Messaging except that the Message passes  
 1558 through multiple intermediate parties. This means that:

- 1559 • the *From Party* (e.g. Party A) and the *To Party* (e.g. Party D) are the only parties that  
 1560 adopt the Reliable Messaging behavior described in this section
- 1561 • the intermediate parties (e.g. Parties B and C), just forward the messages they receive,  
 1562 they do not undertake any Reliable Messaging behavior.

1563 ~~It is RECOMMENDED that Multi-hop Reliable Messaging without Intermediate Acknowledgments~~  
 1564 ~~is used when the From Party that is sending a message is confident that the total time taken for ...~~  
 1565 ~~–the message being acknowledged to be sent to the To Party, and~~  
 1566 ~~–the acknowledgment message to be returned~~

1567 ~~... is sufficiently short so that the From Party will not resend the message being~~  
 1568 ~~acknowledged.<DB>Chris thinks this is superfluous, David thinks it useful as it explains why you~~  
 1569 ~~should do multi-hop and helps an implomenter decide when to use it.</DB>~~

1570 This is described in more detail below:

- 1571 1) The *From Party* and the *To Party* adopt the sending message and receiving message  
 1572 behavior described in sections 10.2.1.1 and 10.2.1.2 except that the *From Party* MSH (e.g.  
 1573 Party A) sends to an Intermediate Party (e.g. Party B) a message (the *message being*  
 1574 *acknowledged*) e.g. Message X in transmission 1, that contains
  - 1575 a) a **QualityOfServiceInfo** element with **deliverySemantics** set to **OnceAndOnlyOnce**
  - 1576 b) a **RoutingHeader** element that contains the **SenderURI** of the sender (e.g. the URL for  
 1577 Party A's MSH) and the **ReceiverURI** of the next recipient of the message (e.g. the URL  
 1578 of Party B's MSH)
- 1579 2) Once the Intermediate Party (e.g. Party B or Party C) receives the message, they determine  
 1580 its next destination (in the example above this could be done by the Routing Application) and  
 1581 forward the message (e.g. Transmission 2 of Message X) to the next Party (e.g. either Party  
 1582 C or Party D). Before sending the message they do the following:
  - 1583 a) transfer elements in the ebXML Header and Payload unchanged from the inbound  
 1584 message to the outbound message except that, they
  - 1585 b) add a **RoutingHeader** element to the **RoutingHeaderList** that contains the **SenderURI**  
 1586 of the next party to receive the message (e.g. the URL for Party C's or Party D's MSH)  
 1587 and the **ReceiverURI** (e.g. the URL for Party B's or Party C's MSH)

1588 3) If the Sending MSH (either at the From Party or at an Intermediate Party) does not receive an  
 1589 Acknowledgment Message after the maximum number of retries, the Sending MSH SHOULD  
 1590 notify the following of the delivery failure:

- 1591 a) The application and/or system administrator function if the Sending MSH is the From  
 1592 Party MSH, or

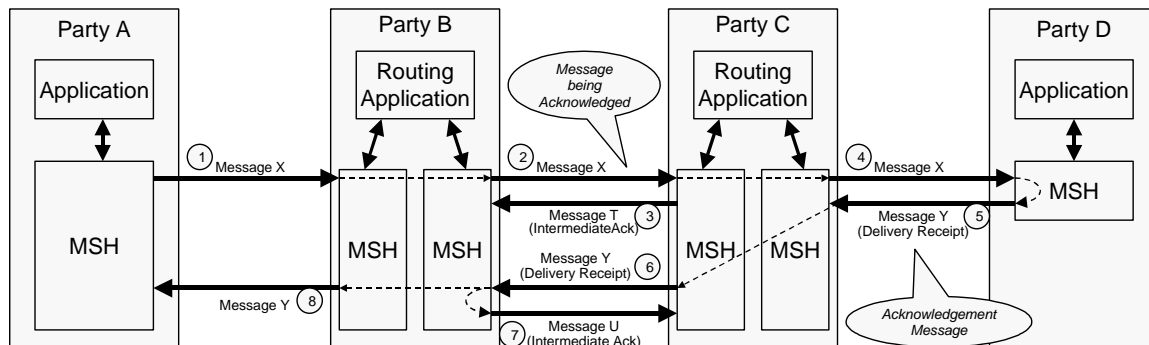
1593 b) The Sending MSH of the From Party, if the Sending MSH is operated by an Intermediate  
 1594 Party (see section 10.5)

- 1595 4) The previous step then repeats until eventually the message (e.g. Message X) reaches its  
 1596 final destination at the To Party (e.g. Party D)
- 1597 5) Once the To Party receives the message (i.e. the message being acknowledged) they return  
 1598 an acknowledgment message to the From Party through the Intermediate Parties.)
- 1599 6) Steps 2 and 3 above then repeat until the acknowledgment message reaches the To Party  
 1600 (e.g. Party A)

1601 **10.2.2.2 Multi-hop Reliable Messaging with Intermediate Acknowledgments**

1602 Multi-hop Reliable Messaging with Intermediate Acknowledgments is similar to Multi-hop Reliable  
 1603 Messaging without Intermediate Acknowledgment except that any of the Parties that are  
 1604 transmitting a Message can request that the recipient return an *Intermediate Acknowledgment*.

1605 This is illustrated by the diagram below.



1606

1607 **Figure 10-6 Multi-hop Reliable Messaging with Intermediate Acknowledgments**

1608 The main difference between Multi-Hop Reliable Messaging with Intermediate Acknowledgments  
 1609 and the without is:

- 1610 • any party may request an intermediate acknowledgment  
 1611 • any party that either sends or receives a message that requests an intermediate  
 1612 acknowledgment must adopt the reliable messaging behavior even if the  
 1613 **QualityOfServiceInfo** element indicates otherwise.

1614 ~~It is RECOMMENDED that Multi-hop Reliable Messaging with Intermediate Acknowledgments is~~  
 1615 ~~used when the From Party that is sending a message is considers that the total time taken for ...~~  
 1616 ~~–the message being acknowledged to be sent to the To Party, and~~  
 1617 ~~–the acknowledgment message to be returned~~

1618 ~~... is so long that the From Party will resend the message being acknowledged. <DB> Should we~~  
 1619 ~~move this to the end of section 10.2.2 intro? <DB>~~

1620 The rules that apply to Multi-hop Reliable Messaging with Intermediate Acknowledgment are as  
 1621 follows:

- 1622 1) Any Party that is sending a message can request that the recipient send an *Acknowledgment*  
 1623 *Message* that is an *Intermediate Acknowledgment* by setting the  
 1624 **IntermediateAckRequested** of the **RoutingHeader** for the hop to **Signed** or **Unsigned**.  
 1625 (e.g. Transmission 2 of Message X, or Transmission 6 of Message Y)
- 1626 2) If a MSH that is not the *To Party* receives a message that requires an Intermediate  
 1627 Acknowledgment (e.g. Transmission 2 of Message X, or Transmission 6 of Message Y) then:

- 1628 a) If the MSH can identify itself as the **ReceiverURI** in the **RoutingHeader** for the hop, and  
 1629 an *Intermediate Acknowledgment* is requested, then the MSH must return an  
 1630 *Acknowledgment Message* (e.g. Transmission 3 of Message T, or Transmission 7 of  
 1631 Message U) with:
  - 1632 i) The **Service** and **Action** elements set as in defined in section 10.4
  - 1633 ii) The **From** element contains the **ReceiverURI** from the last **RoutingHeader** in the  
 1634 message that has just been received
  - 1635 iii) The **To** element contains the **SenderURI** from the last **RoutingHeader** in the  
 1636 message that has just been received
  - 1637 iv) a **RefToMessageId** element that contains the **MessageId** of the message being  
 1638 acknowledged
  - 1639 v) a **QualityOfServiceInfo** element with **deliverySemantics** set to  
 1640 **OnceAndOnlyOnce**
  - 1641 vi) an **Acknowledgment** element with type set to **IntermediateAck**
  - 1642 vii) a **RoutingHeader** element that contains the **SenderURI** of the sender (e.g. the URL  
 1643 for Party C's or Party B's MSH) and the **ReceiverURI** of the next recipient of the  
 1644 message (e.g. the URL of Party B's or Party C's MSH)
- 1645 3) If a MSH that is the *To Party* receives a message and it requires an Intermediate  
 1646 Acknowledgment (see step 2) then, unless the *To Party* is returning an *Acknowledgment*  
 1647 *Message* that is a *Delivery Receipt*, return an *Acknowledgment Message* as described in step  
 1648 2c above.

### 1649 10.3 ebXML Reliable Messaging using Queuing Transports

1650 This section describes the differences that apply if a Queuing Transport is used to implement  
 1651 Reliable Messaging.

1652 Use of the ebXML Reliable Messaging Protocol is identified by the **ReliableMessagingMethod**  
 1653 parameter being set to **Transport** for transmission (either a Single-hop or a Multi-hop)

1654 If Reliable Messaging using a Queuing Transport is being used then the following rules apply:

- 1655 1) An Intermediate Ack SHOULD not be requested. If an Intermediate Ack is requested, then it  
 1656 is ignored.
- 1657 2) No message acknowledgments with an **Acknowledgment** element with a **type** of  
 1658 **IntermediateAck** should be sent, even if requested
- 1659 3) Implementations should use the facilities of the Queuing Transport to determine if the  
 1660 message was delivered
- 1661 4) If an intermediate MSH cannot forward a message to the next Party then the From Party  
 1662 should be notified using the procedure described in section 10.5.
- 1663 5) An acknowledgment message with an **Acknowledgment** element with a type attribute set to  
 1664 **deliveryReceipt** can be sent if requested to inform the sender of the message being  
 1665 acknowledged that the message was delivered.

### 1666 10.4 Service and Action Element Values

1667 An **AcknowledgmentAcknowledgment** element can be included in an **ebXMLHeader** that is  
 1668 part of a *message* that is being sent as a result of processing of an earlier message. In this case  
 1669 the values for the **Service** and **Action** elements are set by the designer of the Service (see  
 1670 section 8.4.4).

1671 An **Acknowledgement** element also can be included in an **ebXMLHeader** that  
 1672 does not include any results from the processing of an earlier message. In this case, the values of  
 1673 the **Service** and **Action** elements MUST be set as follows:

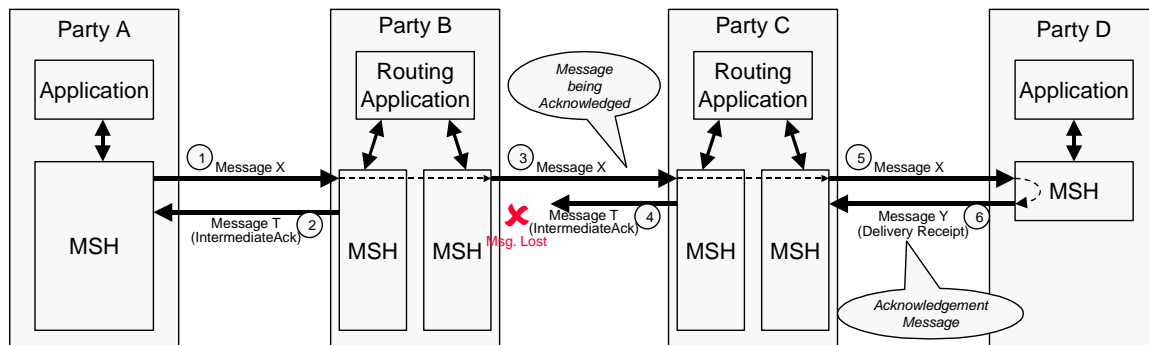
- 1674 • The **Service** element MUST be set to:  
 1675 **[http://www.ebxml.org/namespaces/messageService/MessageAcknowledgementAc  
 1677 knowledgegment](http://www.ebxml.org/namespaces/messageService/MessageAcknowledgementAc<br/>
    1676 knowledgegment)**
- 1677 • The **Action** element MUST be set to the value of the **type** attribute in the  
 1678 **Acknowledgement** element.

1679 Note that **deliveryReceiptRequested** must be set to **None** on a message that is only an  
 1680 **acknowledgement**.

### 1681 10.5 Failed Message Delivery

1682 It is possible, that a Message cannot be delivered to its ultimate destination. This can be either:

- 1683 • when the *To Party* MSH cannot deliver the message to the Application or other process  
 1684 that needs it, or
- 1685 • when using Intermediate Acknowledgments and an Intermediate system determines that  
 1686 a message may have been lost. This is illustrated by the diagram below.



1687

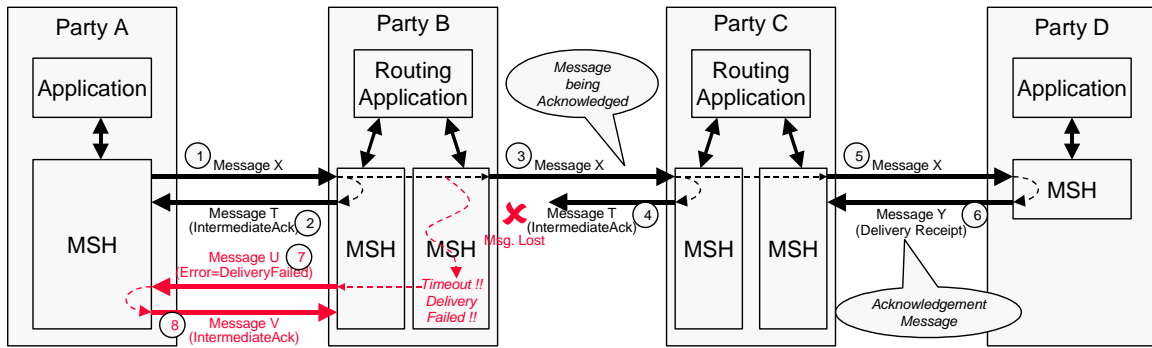
1688 **Figure 10-7 Failed Message Delivery using Intermediate Acknowledgments**

1689 In this example, Party B does not know if Party C (or Party D) has received the message since,  
 1690 even after resending, it has not received the *acknowledgment message* (Message T).

1691 In both these circumstances the MSH that detects the problem MUST send a message to the  
 1692 *From Party* that sent the *message being acknowledged* (via the Intermediate Party if required).  
 1693 The message contains:

- 1694 • a **From Party** that identifies the Party that detected the problem
- 1695 • a **To Party** that identifies the **From Party** that created the message that could not be  
 1696 delivered
- 1697 • a **Service** element and **Action** element set as described in 11.5
- 1698 • a **ReliableMessagingInfoQualityOfServiceInfo** element with **deliverySemantics** set to  
 1699 the same value as the **deliverySemantics** on the message that could not be delivered
- 1700 • an **Error** element with a severity of:
  - 1701 - **Error** if the Party that detected the problem could not even transmit the message  
 1702 (e.g. Transmission 3 was impossible)
  - 1703 - **Warning** if the message (e.g. Message X in Transmission 3) was transmitted, but no  
 1704 acknowledgment was received. This means that the message probably was not  
 1705 delivered although there is a small probability that it was
- 1706 • an **ErrorCode** of **DeliveryFailure**

1707 This is illustrated by the diagram below by the text and arrows in red.



1708

1709 **Figure 10-8 Reporting Failed Message Delivery**

1710 Note that the message that contains an **Error** element with an **ErrorCode** of **DeliveryFailure**  
 1711 (e.g. Message U in Transmission 7) might be sent reliably. It is possible the *acknowledgment*  
 1712 *message* for this message (e.g. Message V in Transmission 8) is not received. In this case, the  
 1713 Party that detects the failed delivery (e.g. Party B) SHOULD inform the Party (e.g. Party A)  
 1714 that sent the *message being acknowledged* (e.g. Message X in Transmission 1) of the failure. How  
 1715 this is done is outside the scope of this specification.

1716 **10.6 Reliable Messaging Parameters**

1717 This section describes the parameters required to control reliable messaging. This parameter  
 1718 information may be contained:

- 1719 • in the ebXML Message header, or
- 1720 • in the CPA associated with the message.

1721 If the information is in both the ebXML message header and the CPA, the information in the  
 1722 header over-rides the CPA.

1723 **10.6.1 Who sets Message Service Parameters**

1724 The values to be used in parameters can be specified by the following parties:

- 1725 • the *From Party*
- 1726 • the *To Party*
- 1727 • the sending Message Service Handler (MSH)
- 1728 • the receiving Message Service Handler

1729 Parameters set by the *From Party* or the *To Party*, apply to the delivery of a message as a whole.  
 1730 Parameters set by the sending or receiving MSH apply to a single-hop.

1731 Note that the *From Party* is the sending MSH and the *To Party* is the receiving MSH for the  
 1732 first/last MSH that handles the message.

1733 The table below indicates where these parameters may be set.

1734

Specified By	Parameter	CPA/ CPP	Message Header	Routing Header
From Party	deliverySemantics	Yes	Yes	N/A
From Party	deliveryReceiptRequested	Yes	Yes	N/A
From Party	syncReplyMode	Yes	Yes	N/A
From Party	timeToLive	Yes	Yes	N/A

Specified By	Parameter	CPA/ CPP	Message Header	Routing Header
To Party	deliveryReceiptProvided	Yes	No	No
Sending MSH	reliableMessagingMethod	No	N/A	Yes
Sending MSH	intermediateAckRequested	No	N/A	Yes
Sending MSH	timeout	Yes	No	No
Sending MSH	retries	Yes	No	No
Sending MSH	retryInterval	Yes	No	No
Receiving MSH	reliableMessagingSupported	Yes	No	No
Receiving MSH	intermediateAckSupported	Yes	No	No
Receiving MSH	persistDuration	Yes	No	No
Receiving MSH	mshTimeAccuracy	Yes	No	No

1735 In this table, the following interpretation of the columns should be used:

1736 ~~4)7)~~ the **Specified By** column indicates the Party that sets the value in the Collaboration Party  
1737 Protocol, Message Header, or Routing Header

1738 ~~2)8)~~ if the **CPA/CPP** column contains a **Yes** then it indicates that the party in the **Specified By**  
1739 column specifies the value that is present in the CPP

1740 ~~3)9)~~ if the **CPA/CPP** column contains a **No** then it indicates that the parameter value is never  
1741 specified in the **CPP**

1742 ~~4)10)~~ if the **Message Header** or **Routing Header** columns contain a **Yes** then it indicates that  
1743 the parameter value may be specified in the **Header** element or **Routing Header** and over-  
1744 rides any value in the CPA. If the value is not specified in the **Header element** or **Routing**  
1745 **Header** then the value in the **CPA** must be used.

1746 ~~5)11)~~ if the **Message Header/Routing Header** columns contain a **No** then it indicates that the  
1747 value in the **CPA** is always used

1748 ~~6)12)~~ if the **Message Header/Routing Header** columns contain a **N/A** then it indicates that the  
1749 value may be specified in another header

1750 These parameters are described below.

## 1751 10.6.2 From Party Parameters

1752 This section describes the parameters that are set by the *From Party*

### 1753 10.6.2.1 Delivery Semantics

1754 The **deliverySemantics** parameter may be present as either an element within the  
1755 **ebXMLHeader** element or as a parameter within the CPA. See section 8.4.7.1 for more  
1756 information.

### 1757 10.6.2.2 Delivery Receipt Requested

1758 The **deliveryReceiptRequested** parameter may be present as either an element within the  
1759 **ebXMLHeader** element or as a parameter within the CPA. See section 8.4.7.2 for more  
1760 information.

### 1761 10.6.2.3 Sync Reply Mode

1762 The **syncReplyMode** parameter may be present as either an element within the **ebXMLHeader**  
1763 element or as a parameter within the CPA. See section 8.4.7.3 for more information.

#### 1764 10.6.2.4 Time To Live

1765 The *TimeToLive* element may be presented within the *ebXMLHeader* element see section  
1766 8.4.7.2 for more information.

#### 1767 10.6.3 To Party Parameters

1768 This section describes the parameters that are set by the *To Party*

##### 1769 10.6.3.1 Delivery Receipt Provided

1770 The *DeliveryReceiptProvided* parameter indicates whether a *To Party* can provide an  
1771 *acknowledgment message* with a *type* attribute of *deliveryReceipt* in response to a message.  
1772 Valid values are:

- 1773 • **Signed** - indicates that only a signed Delivery Receipt can be provided
- 1774 • **Unsigned** - indicates only an unsigned Delivery Receipt can be provided,
- 1775 • **Both** - indicates that either a signed or an unsigned Delivery Receipt can be provided, or
- 1776 • **None** - indicates that the *To Party* does not create Delivery Receipts

1777 If a MSH receives a Message where *deliveryReceiptRequested* is in not compatible with the  
1778 value of *DeliveryReceiptProvided* then the MSH MUST return an *Error Message* to the *From*  
1779 *Party* MSH, reporting that the *DeliveryReceiptProvided* is not supported. This must contain an  
1780 *errorCode* set to **NotSupported** and a *severity* of Error.

#### 1781 10.6.4 Sending MSH Parameters

1782 This section describes the parameters that are set by the *Party* that operates the Sending MSH.

##### 1783 10.6.4.1 Reliable Messaging Method

1784 The *ReliableMessagingMethod* parameter indicates the requested method for Reliable  
1785 Messaging that will be used when sending a Message. Valid values are:

- 1786 • **ebXML** in this case the ebXML Reliable Messaging Protocol as defined in section 10.2 is  
1787 followed, or
- 1788 • **Transport**, in this case a Queuing Transport Protocol is used for reliable delivery of the  
1789 message, see section [040.3](#).

##### 1790 10.6.4.2 Intermediate Ack Requested

1791 The *IntermediateAckRequested* parameter is used by the Sending MSH to request that the  
1792 Receiving MSH that receives the *Message* returns an *acknowledgment message* with an  
1793 *Acknowledgment* element with a *type* of *IntermediateAcknowledgment*.

1794 Valid values for *IntermediateAckRequested* are:

- 1795 • **Unsigned** - requests that an unsigned Delivery Receipt is requested
- 1796 • **Signed** - requests that a signed Delivery Receipt is requested, or
- 1797 • **None** - indicates that no Delivery Receipt is requested.

1798 The default value is **None**.

##### 1799 10.6.4.3 Timeout Parameter

1800 The *timeout* parameter is an integer value that specifies the time in seconds that the Sending  
1801 MSH MUST wait for an *Acknowledgment Message* before first resending a message to the  
1802 Receiving MSH.

1803 **10.6.4.4 Retries Parameter**

1804 The **retries** Parameter is an integer value that specifies the maximum number of times the  
 1805 *message being acknowledged* must be resent to the Receiving MSH using the same  
 1806 Communications Protocol by the Sending MSH.

1807 **10.6.4.5 RetryInterval Parameter**

1808 The **retryInterval** parameter is an integer value specifying, in seconds, the time the Sending  
 1809 MSH MUST wait between retries, if an *Acknowledgment Message* is not received.

1810 **10.6.4.6 Deciding when to resend a message**

1811 The Sending MSH MUST resend the original message if an *Acknowledgment Message* has not  
 1812 been received from the Receiving MSH and either:

- 1813 • the message has not yet been resent and at least the time specified in the **timeout**  
 1814 parameter has passed since the first message was sent, or
- 1815 • the message has been resent, and
  - 1816 - at least the time specified in the **retryInterval** has passed since the last time the  
 1817 message was resent, and
  - 1818 - the message has been resent less than the number of times specified in the **retries**  
 1819 Parameter, and

1820 If the Sending MSH does not receive an *Acknowledgment Message* after the maximum number  
 1821 of retries, the Sending MSH SHOULD notify either:

- 1822 • the application and/or system administrator function if the Sending MSH is the *From*  
 1823 *Party* MSH, or
- 1824 • send an message reporting the delivery failure, if the Sending MSH is operating by an  
 1825 Intermediate Party (see section 10.5)

1826 **10.6.5 Receiving MSH Parameters**

1827 This section describes the parameters that are set by the *Party* that operates the Receiving MSH.

1828 **10.6.5.1 Reliable Messaging Methods Supported**

1829 The **reliableMessagingMethodsSupported** parameter is a list of the methods that a MSH uses  
 1830 to support Reliable Messaging. It must be a URI. The URI for the ebXML Reliable Messaging  
 1831 Protocol described in section 10.2 is <http://www.ebxml.org/namespaces/reliableMessaging>

1832 **10.6.5.2 PersistDuration**

1833 ~~**PersistDuration** is the minimum length of time in days that a Message that is sent reliably is kept  
 1834 in Persistent Storage by a MSH. The value used for **PersistDuration** is an implementation  
 1835 decision although it MUST be greater than the value of the **TimeToLive** parameter for any  
 1836 message that is sent.~~

1837 ~~If a duplicate message (i.e. with the same **MessageId**) is received before the **PersistDuration**  
 1838 has passed, then the MSH that receives it MUST process it as a duplicate message as described  
 1839 in sections 10.2.1.3 and **Error! Reference source not found.**~~

1840 ~~If a duplicate message is received after the **PersistDuration** has passed, then although it may be  
 1841 treated as a duplicate, the sender must realize that it will probably be treated by the MSH as if the  
 1842 message were a new message that had not been received before. **persistDuration** is the  
 1843 minimum length of time, expressed as a [XMLSchema] timeDuration, that data from a *Message*  
 1844 that is sent reliably, is kept in *Persistent Storage* by a MSH that receives that *Message*.~~

1845 In order to support the filtering of duplicate messages, a Receiving MSH MUST, as a minimum,  
1846 save the **MessageId** in *persistent storage*. It is also RECOMMENDED that the following be kept  
1847 in *Persistent Storage*:

- 1848 • the complete message, at least until the information in the message has been passed to  
1849 the application or other process that needs to process it
- 1850 • the time the message was received, so that the information can be used to generate the  
1851 response to a Message Status Request (see section [9.1.1](#))

1852 ***persistDuration*** is specified in the CPA.

1853 A MSH SHOULD NOT resend a message with the same **MessageId** to a receiving MSH if the  
1854 elapsed time indicated by ***persistDuration*** has passed since the message was first sent as the  
1855 receiving MSH will probably not treat it as a duplicate.

1856 If a message cannot be sent successfully before ***persistDuration*** has passed, then the MSH  
1857 should report a delivery failure (see section [10.5](#)).

1858 Note that implementations may determine that a message is persisted for longer than the time  
1859 specified in ***persistDuration***, for example in order to meet legal requirements or the needs of a  
1860 business process. This information is recorded separately within the CPA.

1861 In order to ensure that persistence is continuous as the message is passed from the receiving  
1862 MSH to the process or application that is to handle it, it is RECOMMENDED that a message is  
1863 not removed from *persistent storage* until the MSH knows that the data in the message has been  
1864 received by the process/application.

### 1865 **10.6.5.3 MSH Time Accuracy**

1866 The ***mshTimeAccuracy*** parameter in the CPA indicates the minimum accuracy that a Receiving  
1867 MSH keeps the clocks it uses when checking, for example, ***TimeToLive***. It's value is in the format  
1868 "mm:ss" which indicates the accuracy in minutes and seconds.

## 1869 11 Error Reporting and Handling

1870 This section describes how one ebXML Message Service Handler (MSH) reports errors it detects  
1871 in an ebXML Message to another MSH.

### 1872 11.1 Definitions

1873 For clarity two phrases are defined that are used in this section:

- 1874 • *message in error*. A message that contains or causes an error of some kind
- 1875 • *message reporting the error*. A message that contains an ebXML **ErrorList element** that  
1876 describes the error(s) found in a *message in error*.

### 1877 11.2 Types of Errors

1878 One MSH needs to report to another MSH errors in a *message in error* that are associated with:

- 1879 • the structure or content of the *Message Envelope* (e.g. MIME) (see section 7),
- 1880 • the ebXML Message Header document (see section 8),
- 1881 • reliable messaging failures (see section 10), or
- 1882 • security (see section 12).

1883 Unless specified to the contrary, all references to "an error" in the remainder of this specification  
1884 imply any or all of the types of errors listed above.

1885 Errors associated with Data Communication protocols are detected and reported using the  
1886 standard mechanisms supported by that data communication protocol and are do not use the  
1887 error reporting mechanism described here.

### 1888 11.3 When to generate Error Messages

1889 When an MSH detects an error in a *message in error*, a *message reporting the error* MUST be  
1890 generated and delivered to the MSH that sent the *message in error* if:

- 1891 • the Error Reporting Location (see section 11.4) to which the *message reporting the error*  
1892 should be sent can be determined, and
- 1893 • the *message in error* does not have an **ErrorList** element with **highestSeverity** set to  
1894 **Error**.

1895 If the Error Reporting Location cannot be found or the *message in error* has an **ErrorList** element  
1896 with **highestSeverity** set to **Error**, it is RECOMMENDED that:

- 1897 • the error is logged,
- 1898 • the problem is resolved by other means, and
- 1899 • no further action is taken.

#### 1900 11.3.1 Security Considerations

1901 Party's that receive a Message that contains an error in the header SHOULD always respond to  
1902 the message. However they MAY ignore the message and not respond if they consider that the  
1903 message received is unauthorized or is part of some security attack. The decision process that  
1904 results in this course of action is implementation dependent.

### 1905 11.4 Identifying the Error Reporting Location

1906 The Error Reporting Location is a URI that is specified by the sender of the *message in error* that  
1907 indicates where to send a *message reporting the error*. This may be specified:

- 1908       • by reference, for example by using the **CPAId** to identify the Party Agreement that  
1909       contains the Error Reporting Location, or
- 1910       • by value, for example by using the **ErrorURI** contained within the **RoutingHeader**  
1911       element.
- 1912    If a *message* contains an **ErrorURI** then the **ErrorURI** MUST be used.
- 1913    If an **ErrorURI** is not used then the **ErrorURI** implied by the CPA identified by the **CpaID** on the  
1914    message SHOULD be used. If no **ErrorURI** is implied by the CPA, then the **SenderURI** MUST be  
1915    used.
- 1916    Even if the *message in error* cannot be successfully analyzed or parsed, MSH implementers  
1917    SHOULD try to determine the Error Reporting Location by other means. How this is done is an  
1918    implementation decision.

## 1919    **11.5 Service and Action Element Values**

- 1920    An **ErrorList** element can be included in an **ebXMLHeader** that is part of a *message* that is being  
1921    sent as a result of processing of an earlier message. In this case the values for the **Service** and  
1922    **Action** elements are set by the designer of the Service (see section 8.4.4).
- 1923    An **ErrorList** element can also be included in an **ebXMLHeader** that is not being sent as a result  
1924    of the processing of an earlier message. In this case, the values of the **Service** and **Action**  
1925    elements MUST be set as follows:
- 1926       • The **Service** element MUST be set to:  
1927        **<http://www.ebxml.org/namespaces/messageService/MessageStatus>**
- 1928       • The **Action** element MUST be set to **MessageError**.

## 1929 **12 Security**

1930 [The ebXML Message Service, by its very nature, presents certain security risks. A Message](#)  
1931 [Service may be at risk by means of:](#)

- 1932 [• Unauthorized access](#)
- 1933 [• Data integrity and/or confidentiality attacks \(e.g. through man-in-the-middle attacks\)](#)
- 1934 [• Denial-of-Service, spoofing, bombing attacks](#)

1935 [Each security risk is described in detail in the ebXML Technical Architecture Security](#)  
1936 [Specification \[EBXMLSEC\].](#)

1937 [Each of these security risks MAY be addressed in whole, or in part, by the application of one, or a](#)  
1938 [combination, of the countermeasures described in this section. This specification describes a set](#)  
1939 [of profiles, or combinations of selected countermeasures, that have been selected to address key](#)  
1940 [risks based upon commonly available technologies. Each of the specified profiles includes a](#)  
1941 [description of the risks that are not addressed.](#)

1942 [Application of countermeasures SHOULD be balanced against an assessment of the inherent](#)  
1943 [risks and the value of the asset\(s\) that might be placed at risk.](#)

### 1944 **12.1 Security and Management**

1945 [No technology, regardless of how advanced it might be, is an adequate substitute to the effective](#)  
1946 [application of security management policies and practices.](#)

1947 [It is STRONGLY RECOMMENDED that the site manager of an ebXML Message Service apply](#)  
1948 [due diligence to the support and maintenance of its; security mechanism, site \(or physical\)](#)  
1949 [security procedures, cryptographic protocols, update implementations and apply fixes as](#)  
1950 [appropriate. \(See <http://www.cert.org/> and <http://ciac.llnl.gov/>\)](#)

### 1951 **12.2 Collaboration Protocol Agreement**

1952 [The configuration of Security for MSHs is specified in the CPA. Three areas of the CPA have](#)  
1953 [security definitions as follows:](#)

- 1954 [• The Document Exchange section addresses security to be applied to the payload of the](#)  
1955 [message. The MSH is not responsible for any security specified at this level but may](#)  
1956 [offer these services to the message sender.](#)
- 1957 [• The Message section addresses security applied to the entire ebXML Document, which](#)  
1958 [includes the header and the payload.](#)
- 1959 [• The Transport section addresses the Transport level. The MSH is not responsible for](#)  
1960 [any security specified at this level.](#)

### 1961 **12.3 Countermeasure Technologies**

#### 1962 **12.3.1 Persistent Digital Signature**

1963 [If signatures are being used to digitally sign an ebXML message then XML Signature \[DSIG\]](#)  
1964 [MUST be used to bind the ebXML Header Document to the ebXML Payload or data elsewhere on](#)  
1965 [the web that relates to the message. It is also strongly RECOMMENDED that XML Signature is](#)  
1966 [used to digitally sign the Payload on its own.](#)

1967 [The only available technology that can be applied to the purpose of digitally signing an ebXML](#)  
1968 [Message \(both the ebXMLHeader and its associated payload objects\) is provided by technology](#)  
1969 [that conforms to the W3C/IETF joint XML Signature specification \[XMLDSIG\]. An XML Signature](#)  
1970 [conforming to this specification can selectively sign portions of an XML document\(s\), permitting](#)

1971 [the documents to be augmented \(new element content added\) while preserving the validity of the](#)  
 1972 [signature\(s\).](#)

1973 [An ebXML Message that requires a digital signature SHALL be signed following the processed](#)  
 1974 [defined in this section of the specification and SHALL be in full compliance with \[XMLDSIG\].](#)

1975 **[12.3.1.1 Signature Generation](#)**

1976 [13\) Create a SignedInfo element with SignatureMethod, CanonicalizationMethod, and](#)  
 1977 [Reference\(s\) elements for the ebXMLHeader document and any required payload objects, as](#)  
 1978 [prescribed by \[XMLDSIG\].](#)

1979 [14\) Canonicalize and then calculate the SignatureValue over SignedInfo based on algorithms](#)  
 1980 [specified in SignedInfo as specified in \[XMLDSIG\].](#)

1981 [15\) Construct the Signature element that includes the SignedInfo, KeyInfo \(RECOMMENDED\),](#)  
 1982 [and SignatureValue elements as specified in \[XMLDSIG\].](#)

1983 [16\) Include the namespace qualified Signature element in the ebXMLHeader document just](#)  
 1984 [signed, following the RoutingHeaderList element.](#)

1985 [The ds:SignedInfo element SHALL be composed of zero or one ds:CanonicalizationMethod](#)  
 1986 [element, the ds:SignatureMethod and one or more ds:Reference elements.](#)

1987 [The ds:CanonicalizationMethod element is defined as OPTIONAL in \[XMLDSIG\], meaning that](#)  
 1988 [the element need not appear in an instance of a ds:SignedInfo element. The default](#)  
 1989 [canonicalization method that is applied to the data to be signed is \[XMLC14N\] in the absence of a](#)  
 1990 [ds:Canonicalization element that specifies otherwise. This default SHALL also serve as the](#)  
 1991 [default canonicalization method for the ebXML Message Service.](#)

1992 [The ds:SignatureMethod element SHALL be present and SHALL have an Algorithm attribute. The](#)  
 1993 [RECOMMENDED value for the Algorithm attribute is:](#)

1994 [http://www.w3.org/2000/02/xmldsig#sha1](#)

1995 [This RECOMMENDED value SHALL be supported by all compliant ebXML Message Service](#)  
 1996 [software implementations.](#)

1997 [The ds:Reference element for the ebXMLHeader document SHALL have an URI attribute value](#)  
 1998 [of "" to provide for the signature to be applied to the document that contains the ds:Signature](#)  
 1999 [element \(the ebXMLHeader document\). The ds:Reference element for the ebXMLHeader](#)  
 2000 [document MAY include a Type attribute that has a value](#)  
 2001 ["http://www.w3.org/2000/02/xmldsig#Object" in accordance with \[XMLDSIG\]. This attribute is](#)  
 2002 [purely informative. It MAY be omitted. Implementations of the ebXML MSH SHALL be prepared](#)  
 2003 [to handle either case. The ds:Reference element MAY include the optional id attribute.](#)

2004 [The ds:Reference element for the ebXMLHeader document SHALL include a child ds:Transform](#)  
 2005 [element that excludes the containing ds:Signature element and all its descendants as well as the](#)  
 2006 [RoutingHeaderList element and all its descendants as these elements are subject to change. The](#)  
 2007 [ds:Transform element SHALL include a child ds:XPath element that has a value of:](#)

2008 `/descendant-or-self::node\(\)\[not\(ancestor-or-self::ds:Signature\[@id='S1'\]\) and not\(ancestor-or-`  
 2009 `self::RoutingHeaderList\)\]`  
 2010

2011 [Each payload object that requires signing SHALL be represented by a ds:Reference element that](#)  
 2012 [SHALL have an URI attribute that resolves to that payload object. This MAY be either the](#)  
 2013 [Content-Id URI of the payload object enveloped in the MIME ebXML Payload Container, or an](#)  
 2014 [URI that matches the Content-Location header of the payload object enveloped in the ebXML](#)  
 2015 [Payload Container, or an URI that resolves to an external payload object that is external to the](#)  
 2016 [ebXML Payload Container. It is STRONGLY RECOMMENDED that the URI attribute value match](#)  
 2017 [the xlink:href URI value of the corresponding Manifest/Reference element for that payload object.](#)  
 2018 [However, this is NOT REQUIRED.](#)

2019 [Example of digitally signed ebXMLHeader document:](#)

```

2020
2021 <?xml version="1.0" encoding="utf-8"?>
2022 <ebXMLHeader
2023   xmlns="http://www.ebxml.org/namespaces/messageHeader"
2024   xmlns:xlink="http://www.w3.org/1999/xlink"
2025   version="1.0">
2026   <Manifest id="Mani01">
2027     <Reference xlink:href="cid://blahblahblah"
2028       xlink:role="http://ebxml.org/gci/invoice">
2029       <Schema version="1.0" location="http://ebxml.org/gci/busdocs/invoice.dtd"/>
2030     </Reference>
2031   </Manifest>
2032   <Header>
2033     ...
2034   </Header>
2035   <RoutingHeaderList>
2036     <RoutingHeader>
2037       ...
2038     </RoutingHeader>
2039   </RoutingHeaderList>
2040   <ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmlds#">
2041     <ds:SignedInfo>
2042       <ds:CanonicalizationMethod Algorithm="http://www.w3.org/TR/2000/WD-xml-c14n-20001011"/>
2043       <ds:SignatureMethod Algorithm="http://www.w3.org/2000/09/xmlds#dsa-shal"/>
2044       <ds:Reference URI=" ">
2045         <ds:Transforms>
2046           <ds:Transform>
2047             <XPath>/descendant-or-self::node() [not(ancestor-or-self::ds:Signature[@id='S1']) and
2048 not(ancestor-or-self::RoutingHeaderList)]</XPath>
2049           </ds:Transform>
2050         </ds:Transforms>
2051         <ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmlds#shal"/>
2052         <ds:DigestValue>...</ds:DigestValue>
2053       </ds:Reference>
2054       <ds:Reference URI="cid://blahblahblah/">
2055         <ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmlds#shal"/>
2056         <ds:DigestValue>...</ds:DigestValue>
2057       </ds:Reference>
2058     </ds:SignedInfo>
2059     <ds:SignatureValue>...</ds:SignatureValue>
2060     <ds:KeyInfo>...</ds:KeyInfo>
2061   </ds:Signature>
2062 </ebXMLHeader>
2063

```

2064 **[12.3.2 Persistent Signed Receipt](#)**

2065 [An ebXML Message that has been digitally signed MAY be acknowledged with a DeliveryReceipt](#)  
 2066 [acknowledgment message that itself is digitally signed in the manner described in the previous](#)  
 2067 [section. The acknowledgment message MUST contain the set of ds:DigestValue elements](#)  
 2068 [contained in the ds:Signature element of the original message within the Acknowledgment](#)  
 2069 [element.](#)

2070 **[12.3.3 Non-persistent Authentication](#)**

2071 [Non-persistent authentication is provided by the communications channel used to transport the](#)  
 2072 [ebXML message. This authentication MAY be either in one direction—from the session initiator to](#)  
 2073 [the receiver—or bi-directional. The specific method will be determined by the communications](#)  
 2074 [protocol used. For instance, the use of a secure network protocol, such as \[TLS\] or \[IPSEC\]](#)  
 2075 [provides the sender of an ebXML Message to authenticate the destination for the TCP/IP](#)  
 2076 [environment.](#)

2077 **[12.3.4 Non-persistent Integrity](#)**

2078 [Use of a secure network protocol such as \[TLS\] or \[IPSEC\] MAY be configured so as to provide](#)  
 2079 [for integrity check CRCs of the packets transmitted via the network connection.](#)

### 2080 **12.3.5 Persistent Confidentiality**

2081 [XMLEncryption is an W3C/IETF joint activity that is actively engaged in the drafting of a](#)  
2082 [specification for the selective encryption of an XML document\(s\). It is anticipated that this](#)  
2083 [specification will be completed within the next year. The ebXML Transport, Routing and](#)  
2084 [Packaging team has identified this technology as the only viable means of providing persistent,](#)  
2085 [selective confidentiality of elements within an ebXML Message including the ebXMLHeader](#)  
2086 [document.](#)

2087 [Confidentiality for ebXML Payloads MAY be provided by functionality possessed by a MSH.](#)  
2088 [However, this specification states that it is not the responsibility of the MSH to provide security for](#)  
2089 [the ebXML Payloads. Payload confidentiality MAY be provided by using XMLEncryption \(when](#)  
2090 [available\) or some other cryptographic process, such as \[S/MIME\], \[S/MIMEV3\], or \[PGP/MIME\],](#)  
2091 [that is bilaterally agreed upon by the parties involved. Since XML Encryption is not currently](#)  
2092 [available, it is RECOMMENDED that \[S/MIME\] encryption methods be used for ebXML Payloads.](#)  
2093 [The XML Encryption standard SHALL be the default encryption method when XML Encryption](#)  
2094 [has achieved W3C Recommendation status.](#)

2095 [Section xx \(TBD\) describes RECOMMENDED bindings for providing persistent confidentiality](#)  
2096 [using MIME-based encryption schemes.](#)

### 2097 **12.3.6 Non-persistent Confidentiality**

2098 [Use of a secure network protocol such as \[TLS\] or \[IPSEC\] provides transient confidentiality of a](#)  
2099 [message as it is transferred between two ebXML MSH nodes.](#)

### 2100 **12.3.7 Persistent Authorization**

2101 [The OASIS Security Services TC is actively engaged in the definition of a specification that](#)  
2102 [provides for the exchange of security credentials, including NameAssertion and Entitlements that](#)  
2103 [is based on \[S2ML\]. Use of technology that is based on this anticipated specification MAY be](#)  
2104 [used to provide persistent authorization for an ebXML Message once it becomes available.](#)  
2105 [ebXML has a formal liaison to this TC. There are also many ebXML member organizations and](#)  
2106 [contributors that are active members of the OASIS Security Services TC such as Sun, IBM,](#)  
2107 [CommerceOne, Cisco and others that are endeavoring to ensure that the specification meets the](#)  
2108 [requirements of providing persistent authorization capabilities for the ebXML Message Service.](#)

### 2109 **12.3.8 Non-persistent Authorization**

2110 [Use of a secure network protocol such as \[TLS\] or \[IPSEC\] MAY be configured to provide for](#)  
2111 [bilateral authentication of certificates prior to establishing a session. This provides for the ability](#)  
2112 [for an ebXML MSH to authenticate the source of a connection that can be used to recognize the](#)  
2113 [source as an authorized source of ebXML Messages.](#)

### 2114 **12.3.9 Trusted Timestamp**

2115 [At the time of this specification, services that offer trusted timestamp capabilities are becoming](#)  
2116 [available. Once these become more widely available, and a standard has been defined for their](#)  
2117 [use and expression, these standards, technologies and services will be evaluated and considered](#)  
2118 [for use in providing this capability.](#)

Present in baseline MSH		<a href="#">Persistent digital signature</a>	<a href="#">Non-persistent authentication</a>	<a href="#">Persistent signed receipt</a>	<a href="#">Non-persistent integrity</a>	<a href="#">Persistent confidentiality</a>	<a href="#">Non-persistent confidentiality</a>	<a href="#">Persistent authorization</a>	<a href="#">Non-persistent authorization</a>	<a href="#">Trusted timestamp</a>	<a href="#">Description of Profile</a>
●	<a href="#">Profile 0</a>										no security services are applied to data
●	<a href="#">Profile 1</a>	●									sending MSH applies XML/DSIG structures to message
	<a href="#">Profile 2</a>		●						●		sending MSH authenticates and receiving MSH validates authorization from communication channel credentials
	<a href="#">Profile 3</a>		●			●					sending MSH authenticates and receiving MSH used secure channel to transmit data
	<a href="#">Profile 4</a>		●	●							sending MSH authenticates, the receiving MSH performs integrity checks using communications protocol
	<a href="#">Profile 5</a>		●								sending MSH authenticates the communication channel only (e.g., SSL 3.0 over TCP/IP)
	<a href="#">Profile 6</a>	●				●					sending MSH applies XML/DSIG structures to message and passes in secure communications channel
	<a href="#">Profile 7</a>	●		●							sending MSH applies XML/DSIG structures to message and receiving MSH returns a signed receipt
	<a href="#">Profile 8</a>	●		●		●					combination of profile 6 and 7
	<a href="#">Profile 9</a>	●							●		Profile 5 with a trusted timestamp applied
	<a href="#">Profile 10</a>	●		●						●	Profile 9 with receiving MSH returning a signed receipt
	<a href="#">Profile 11</a>	●				●				●	Profile 6 with the receiving MSH applying a trusted timestamp
	<a href="#">Profile 12</a>	●		●		●				●	Profile 8 with the receiving MSH applying a trusted timestamp
	<a href="#">Profile 13</a>	●			●						sending MSH applies XML/DSIG structures to message and applies confidentiality structures (XML-Encryption)
	<a href="#">Profile 14</a>	●		●		●					Profile 13 with a signed receipt
	<a href="#">Profile 15</a>	●		●						●	sending MSH applies XML/DSIG structures to message, a trusted timestamp is added to message. receiving MSH returns a signed receipt

	Present in baseline MSH	Persistent digital signature	Non-persistent authentication	Persistent signed receipt	Non-persistent integrity	Persistent confidentiality	Non-persistent confidentiality	Persistent authorization	Non-persistent authorization	Trusted timestamp	Description of Profile
											message, receiving MSH returns a signed receipt
	<a href="#">Profile 16</a>	●				●				●	<a href="#">Profile 13 with a trusted timestamp applied</a>
	<a href="#">Profile 17</a>	●		●		●				●	<a href="#">Profile 14 with a trusted timestamp applied</a>
	<a href="#">Profile 18</a>	●						●			<a href="#">sending MSH applies XML/DSIG structures to message and forwards authorization credentials (S2ML)</a>
	<a href="#">Profile 19</a>	●		●				●			<a href="#">Profile 18 with receiving MSH returning a signed receipt</a>
	<a href="#">Profile 20</a>	●		●				●		●	<a href="#">Profile 19 with the a trusted timestamp being applied to the sending MSH message</a>
	<a href="#">Profile 21</a>	●		●		●		●		●	<a href="#">Profile 19 with the sending MSH applying confidentiality structures (XML-Encryption)</a>
	<a href="#">Profile 22</a>					●					<a href="#">sending MSH encapsulates the message within confidentiality structures (XML-Encryption)</a>

2119 The ebXML Message Service, by its very nature, presents certain security risks. A Message  
 2120 Service may be at risk by means of:

- 2121 ~~–Unauthorized access~~
- 2122 ~~–Data integrity and/or confidentiality attacks (e.g. through man-in-the-middle attacks)~~
- 2123 ~~–Denial-of-Service, spoofing, bombing attacks~~

2124 Each of these security risks MAY be addressed in whole, or in part, by the application of one, or a  
 2125 combination, of the countermeasures described in this section. This specification describes a set  
 2126 of profiles, or combinations of selected countermeasures, that have been selected to address key  
 2127 risks based upon commonly available technologies. Each of the specified profiles includes a  
 2128 description of the risks that are not addressed.

2129 Application of countermeasures SHOULD be balanced against an assessment of the inherent  
 2130 risks and the value of the asset(s) that might be placed at risk. <CF> need some reference to risk  
 2131 assessment sites/docs here.</CF>

2132 **12.1 Security and Management**

2133 No technology, regardless of how advanced it might be, is an adequate substitute to the effective  
 2134 application of security management policies and practices.

2135 It is ~~STRONGLY RECOMMENDED~~ that the site manager of an ebXML Message Service apply  
2136 due diligence to the support and maintenance of its; security mechanism, site (or physical)  
2137 security procedures, cryptographic protocols, update implementations and apply fixes as  
2138 appropriate. (See <http://www.cert.org/>, <http://ciac.llnl.gov/>)

## 2139 **12.2 Collaboration Party Profiles**

2140 The configuration of Security for MSHs is specified in the CPP. There are three areas of the CPP  
2141 that have security definitions as follows:

- 2142 ~~–The Document Exchange section addresses security to be applied to the payload of the~~  
2143 ~~message. The MSH is not responsible for any security specified at this level but may~~  
2144 ~~offer these services to the message sender.~~
- 2145 ~~–The Message section addresses security applied to the entire ebXML Document, which~~  
2146 ~~includes the header and the payload.~~
- 2147 ~~–The Transport section addresses the Transport level. The MSH is not responsible for any~~  
2148 ~~security specified at this level.~~

## 2149 **12.3 Risks**

### 2150 **12.3.1 Unauthorized Access**

2151 One of the risks for Message Service Handlers is sending messages to or receiving messages  
2152 from another message service handler that is not known or one that is being impersonated by a  
2153 rogue MSH. Receiving a flood of requests from a known or unknown MSH can be considered a  
2154 denial of service attack. Message Service Handlers need to be identified and need to be able to  
2155 authenticate requests from other MSHs.

2156 A message ~~MAY~~ carry information that authenticates the sending MSH in the message header. If  
2157 authentication data is in the header, it ~~MUST~~ be protected from modification and inappropriate  
2158 access.

2159 Messages that are digitally signed ~~MAY~~ also be asserted as authenticated requests.

2160 Authentication ~~MAY~~ also be provided by the underlying transport, such as through the use of  
2161 [TLS].

### 2162 **12.3.2 Data Integrity and Confidentiality**

2163 Integrity protection is the term used to express a requirement that data ~~MUST~~ be protected from  
2164 unauthorized modification while it is stored or passed over the network. The common technology  
2165 for integrity protection is to generate a hash of the data and storing both the information and the  
2166 hash securely. In a network protocol the hash is sent through a protected means and ~~MAY~~ be  
2167 used to validate that the data received is the same data that was sent.

2168 Privacy, or confidentiality, is the term used to express a requirement that data ~~MUST~~ be  
2169 encrypted while it is stored or passed over the network. The common method for privacy  
2170 protection is encryption via symmetric key algorithm like DES or triple-DES.

### 2171 **12.3.3 Denial of Service**

2172 It is assumed that ebXML data and operations flow over the existing web infrastructure. All  
2173 message services will implement their own web security infrastructure and practices. There are  
2174 threats at all levels of the stack that need to be addressed through other means outside ebXML.  
2175

## 2176 **12.4 Countermeasure Technologies**

### 2177 **12.4.1 ebXML Message Countermeasures for Unauthorized Access and Data** 2178 **Integrity**

### 2179 **12.4.2 Digital Certificates**

2180 The X.509 v3 standard describes an extensible framework within which basic certificate  
2181 information MAY be extended. It also describes how such extensions MAY be used to control the  
2182 process of issuing and validating certificates. Presently, there is no single view as to which  
2183 certificate extensions must be present in an X.509 v3 digital certificate. The Collaboration  
2184 Protocol Agreement identifies the particular X.509 v3 certificate extensions that the parties to an  
2185 agreement have agreed to use. An implementation of the ebXML Message Service MAY handle  
2186 the subset of the certificate extensions listed in [S/MIME], but this capability is NOT REQUIRED.  
2187 A Message Service that receives a message that contains critical extensions in an X509 v3  
2188 certificate that it is unable to handle MUST abandon processing of the message and return an  
2189 Error (see section 11 with **errorCode** set to **SecurityFailure** and **severity** set to **Error**).

2190 An implementation of the ebXML Message Service MAY implement a certificate revocation list  
2191 (CRL) retrieval mechanism. The purpose of a CRL is to gain access to certificate revocation  
2192 information when validating certificate chains. It is RECOMMENDED that the Message Service  
2193 retrieve and utilize CRL information each time a certificate is verified. The ultimate decision  
2194 regarding use of the CRL information is left to the security policy of a party deploying an ebXML  
2195 Message Service.

2196 The use of a digital signature on an ebXML message satisfies the requirements for message  
2197 integrity verification as well as authentication of the sender's identity. The digital signature also  
2198 helps to establish the ebXML message non-repudiation property.

2199

### 2200 **12.4.3 ebXML Message Countermeasures for Denial of Service**

2201 Message Service implementations SHOULD be able to immediately detect messages that MAY  
2202 be a denial of service attack and take appropriate measures to reject these messages. Message  
2203 Service implementations SHOULD be able to authenticate the claimed identity of a message  
2204 sender when authentication is required by the business. *<MH> need to tie this in to requirements  
2205 in the CPP for authenticating at the MSH or transport level. </MH>*

### 2206 **12.4.4 ebXML Management Countermeasures for Denial of Service**

2207 It is STRONGLY RECOMMENDED that the site manager of an ebXML Message Service take  
2208 appropriate measures to monitor announcements and descriptions of new attacks (See  
2209 <http://www.cert.org/>) and apply updates and patches as appropriate.

## 2210 **12.5 Profiles**

### 2211 **12.5.1 XML Digital Signature (XMLDSIG)**

2212 The joint W3C/IETF XMLDSIG Working Group has released the [XMLDSIG] specification as a  
2213 Candidate Recommendation effective November, 2000. This means that the specification is  
2214 made public for the purposes of encouraging implementations of the specification to validate that  
2215 it can be successfully implemented. To date, there are at least three implementations of the  
2216 specification in circulation, with others under development. It is anticipated that this specification,  
2217 along with the recently initiated XML Encryption Working Group (also a joint W3C/IETF initiative)  
2218 will be key technologies that MAY be employed by the ebXML Message Service.

2219 The [XMLDSIG] specification defines how an XML document(s) MAY be signed, either in whole  
2220 or as selective element content by means of a transformation such as [XPath] or [XSLT].

2221 **12.5.2 Profile – XML Signature signing of header and/or payload**

2222 An ebXML Message MAY be signed using technology that implements the [XMLDSIG]  
2223 specification.

2224 **To be completed.**

2225 **12.5.2.1 Risks**

2226 This profile does not provide persistent privacy/confidentiality. It is STRONGLY RECOMMENDED  
2227 that this profile be used in conjunction with a secure transport that provides for authentication as  
2228 well as encryption over the network such as is provided by [TLS]. HTTP over SSL (HTTP/S)  
2229 would be such a transport mechanism.

2230 **12.5.2.2 Benefits**

2231 This profile provides the only means of signing both the header and payload objects. This profile  
2232 also allows the message to be modified as it traverses through intermediary Message Service  
2233 Handlers that MUST append **RoutingHeader** elements as the message is (re) sent on its path  
2234 from the From Party to the To Party.

2235 **12.5.3 S/MIME**

2236 [S/MIME] names the message digest algorithms (md5, sha1), the public key encryption algorithm  
2237 (RSA), and the bulk data encryption algorithms (RC2/40 and, optionally, Triple DES) that MUST  
2238 be implemented in order to comply with the standard. An implementation of the ebXML Message  
2239 Service that claims support for S/MIME SHALL conform to that standard.

2240 The [S/MIME] specification REQUIRES that each MIME entity to be signed and/or encrypted  
2241 MUST be converted to a canonical form that may be uniquely and unambiguously represented in  
2242 both the environment where the signature is to be created and the environment where the  
2243 signature is to be verified. MIME entities MUST be presented in a canonical format for enveloping  
2244 as well as for signing.

2245 The S/MIME specification RECOMMENDS transmitting entities such as 8-bit text and binary data  
2246 to be encoded with quoted-printable or base-64 transfer encoding. This provision applies to  
2247 formatting of the ebXML messages due to the transport independence property of the protocol.

2248 Digital certificates are delivered as a part of the application/pkcs7-signature part of the  
2249 multipart/signed message. [S/MIMECH] provides the guidelines for use of the digital certificates in  
2250 S/MIME messages. The exact implementation of the certificate handling procedures and  
2251 authentication semantics of the information in the digital certificate received with an ebXML  
2252 message is left to the Trading Partner Agreement. <CF> this needs work! </CF>

2253 **12.5.4 Profile – S/MIME signing of message payload**

2254 The multipart/signed form defined by the [S/MIME] specification MAY be used to sign ebXML  
2255 message payloads. This specification makes no claims as to how the signing and packaging of  
2256 the payload object(s) is to be achieved. An implementation of the ebXML Message Service MAY  
2257 choose to offer these services to the application or application service layers of software as  
2258 described in the section on the Message Service Interface. However, this is not a REQUIRED  
2259 feature of an ebXML Message Service.

2260 This profile SHALL be uniquely identified by the following URI:

2261 – <http://www.ebxml.org/namespaces/security-profiles/smime-pkcs7-signed-payload>

2262 The [S/MIME] specification REQUIRES two parameters of the multipart/signed content type:

2263 – protocol

2264 – micalg

2265 An ebXML message payload that is signed using this profile SHALL use the following values for  
 2266 these MIME parameters:  
 2267     ~~–protocol="application/pkcs7-signature~~  
 2268     ~~–micalg="rsa-sha1"~~

2269 **12.5.4.1 Sample S/MIME signed payload**

2270 The following is a sample S/MIME signed payload

```

2271 Content-Type: multipart/related; type="application/vnd.eb+xml; version="0.91"; version="0.92";
2272 boundary=ebxmlenvelopeuniquestring;
2273 Content-Id: localpart@domain
2274
2275 --ebxmlenvelopeuniquestring
2276 Content-Type: application/vnd.eb+xml; version="0.91"; version="0.92"; charset="UTF-8";
2277 Content-Id: localpart@domain
2278
2279 <?xml version="1.0" encoding="UTF-8"?>
2280 <ebXMLHeader version="0.91" version="0.92" xmlns=http://www.ebxml.org/namespaces/messageHeader>
2281 ==
2282 </ebXMLHeader>
2283
2284 --ebxmlenvelopeuniquestring
2285
2286 Content-Type: multipart/signed; boundary="someuniquestring"; protocol="application/pkcs7-
2287 signature"; micalg="rsa-sha1";
2288 Content-Id: localpart@domain
2289
2290 --someuniquestring
2291
2292 Content-Type: text/plain
2293 Content-Id: localpart@domain
2294
2295 <Payload in the clear>
2296
2297 --someuniquestring
2298 Content-Type: application/pkcs7-signed; name="smime.p7s";
2299 Content-Id: localpart@domain
2300 Content-Transfer-Encoding: base64
2301
2302 %^)*&TLVGSRKWHF
2303
2304 --someuniquestring--
2305 --ebxmlenvelopeuniquestring--
    
```

2306 **12.5.4.2 Risks**

2307 This profile does not provide persistent privacy/confidentiality. It is STRONGLY RECOMMENDED  
 2308 that this profile be used in conjunction with a secure transport that provides for authentication as  
 2309 well as encryption over the network such as is provided by [TLS]. HTTP over SSL (HTTP/S)  
 2310 would be such a transport mechanism.

2311 The header document is unsigned and there is no binding of the header and payload.

2312 **12.5.4.3 Benefits**

2313 This is the simplest form of integrity, with application signing and authentication of the payload  
 2314 only.

2315 **12.5.5 Profile – S/MIME encryption of message payload**

2316 This profile SHALL be uniquely identified by the following URI:

2317     ~~–http://www.ebxml.org/namespaces/security-profiles/smime-pkcs7-encrypted-payload~~

2318 The [S/MIME] specification REQUIRES two parameters of the multipart/signed content type:

2319     ~~–protocol~~

- 2320        ~~—micalg~~
- 2321        ~~An ebXML message payload that is signed using this profile SHALL use the following values for~~  
2322        ~~these MIME parameters:~~
- 2323        ~~—protocol="application/pkcs7-signature~~  
2324        ~~—micalg="rsa-sha1"~~
- 2325        **12.5.5.1 Risks**
- 2326        ~~The header document is unsigned and there is no binding of the header and payload.~~
- 2327        **12.5.5.2 Benefits**
- 2328        ~~This is the simplest form of integrity, with application signing and authentication of the payload~~  
2329        ~~only.~~
- 2330        **12.5.6 PGP/MIME**
- 2331        ~~[PGP/MIME] MAY be used to sign and/or encrypt an ebXML message payload object(s). An~~  
2332        ~~implementation of the ebXML Message Service that claims support for PGP/MIME SHALL~~  
2333        ~~conform to that standard.~~
- 2334        **12.5.7 Profile – PGP/MIME signing of message payload**
- 2335        ~~TBD – Dick~~
- 2336        **12.5.7.1 Risks**
- 2337        ~~This profile does not provide persistent privacy/confidentiality. It is STRONGLY RECOMMENDED~~  
2338        ~~that this profile be used in conjunction with a secure transport that provides for authentication as~~  
2339        ~~well as encryption over the network such as is provided by [TLS]. HTTP over SSL (HTTP/S)~~  
2340        ~~would be such a transport mechanism.~~
- 2341        ~~The header document is unsigned and there is no binding of the header and payload.~~
- 2342        **12.5.7.2 Benefits**
- 2343        **12.5.8 Profile – PGP/MIME encryption of message payload**
- 2344        ~~TBD – Dick~~
- 2345        **12.5.8.1 Risks**
- 2346        **12.5.8.2 Benefits**

2347 **13 Synchronous and Asynchronous Responses**

2348 This section may not be needed.

## 2349 14 References

2350 <DB>What's the difference between normative and non-normative</DB>

### 2351 14.1 Normative References

- 2352 [HTTP] RFC 2068 - Hypertext Transfer Protocol -- HTTP/1.1, R. Fielding, J. Gettys,  
2353 J. Mogul, H. Frystyk, T. Berners-Lee, January 1997
- 2354 [ISO 8601] International Standards Organization Ref. ISO 8601 Second Edition,  
2355 Published 1997
- 2356 [RFC 2392] IETF Request For Comments 2392. Content-ID and Message-ID Uniform  
2357 Resource Locators. E. Levinson, Published August 1998
- 2358 [RFC 2396]
- 2359 [RFC2045] IETF RFC 2045. Multipurpose Internet Mail Extensions (MIME) Part One:  
2360 Format of Internet Message Bodies, N Freed & N Borenstein, Published  
2361 November 1996
- 2362 [SMTP] RFC 821, Simple Mail Transfer Protocol, J Postel, August 1982
- 2363 [TLS] RFC2246, T. Dierks, C. Allen. January 1999.
- 2364 [UTF-8] UTF-8 is an encoding that conforms to ISO/IEC 10646. See [XML] for usage  
2365 conventions.
- 2366 [XML] W3C XML 1.0 Recommendation,  
2367 <http://www.w3.org/TR/2000/REC-xml-20001006>
- 2368 [XML Namespace] Recommendation for Namespaces in XML, World Wide Web Consortium, 14  
2369 January 1999, <http://www.w3.org/TR/REC-xml-names>

### 2370 14.2 Non-Normative References

- 2371 [Glossary] ebXML Glossary, see ebXML Project Team Home Page
- 2372 [PGP/MIME] RFC2015, "MIME Security with Pretty Good Privacy (PGP)", M. Elkins.  
2373 October 1996.
- 2374 [S/MIME] RFC2311, "S/MIME Version 2 Message Specification", S. Dusse, P.  
2375 Hoffman, B. Ramsdell, L. Lundblade, L. Repka. March 1998.
- 2376 [S/MIMECH] RFC 2312, "S/MIME Version 2 Certificate Handling", S. Dusse, P. Hoffman,  
2377 B. Ramsdell, J. Weinstein. March 1998.
- 2378 [TRPREQ] ebXML Transport, Routing and Packaging: Overview and Requirements,  
2379 Version 0.96, Published 25 May 2000
- 2380 [XLINK] W3C Xlink Candidate Recommendation, <http://www.w3.org/TR/xlink/>
- 2381 [XMLDSIG] Joint W3C/IETF XML Digital Signature specification,  
2382 <http://www.w3.org/TR/2000/CR-xmlsig-core-20001031/>
- 2383 [XMLMedia] IETF Internet Draft on XML Media Types. See [http://www.imc.org/draft-](http://www.imc.org/draft-murata-xml-08)  
2384 [murata-xml-08](http://www.imc.org/draft-murata-xml-08). Note. It is anticipated that this Internet Draft will soon become  
2385 a RFC. Final versions of this specification will refer to the equivalent RFC.
- 2386 [XMLSchema] W3C XML Schema Candidate Recommendation,  
2387 <http://www.w3.org/TR/xmlschema-0/>  
2388 <http://www.w3.org/TR/xmlschema-1/>  
2389 <http://www.w3.org/TR/xmlschema-2/>

2390 [XMTP] XMTP - Extensible Mail Transport Protocol  
2391 <http://www.openhealth.org/documents/xmtp.htm>

2392 **15 Disclaimer**

2393 The views and specification expressed in this document are those of the authors and are not  
2394 necessarily those of their employers. The authors and their employers specifically disclaim  
2395 responsibility for any problems arising from correct or incorrect implementation or use of this  
2396 design.

2397 **16 Contact Information**

2398 **Team Leader**

2399 Name Rik Drummond  
 2400 Company Drummond Group, Inc.  
 2401 Street 5008 Bentwood Crt.  
 2402 City, State, Postal Code Fort Worth, Texas 76132  
 2403 Country USA  
 2404 Phone +1 (817) 294-7339  
 2405 EMail: rik@drummondgroup.com

2406  
 2407 **Vice Team Leader**

2408 Name Chris Ferris  
 2409 Company Sun Microsystems  
 2410 Street One Network Drive  
 2411 City, State, Postal Code Burlington, MA 01803-0903  
 2412 Country USA  
 2413 Phone: +1 (781) 442-3063  
 2414 EMail: chris.ferris@sun.com

2415  
 2416 **Team Editor**

2417 Name David Burdett  
 2418 Company Commerce One  
 2419 Street 4400 Rosewood Drive  
 2420 City, State, Postal Code Pleasanton, CA 94588  
 2421 Country USA  
 2422 Phone: +1 (925) 520-4422  
 2423 EMail: david.burdett@commerceone.com

2424  
 2425 **Authors**

2426 Name Dick Brooks  
 2427 Company Group 8760  
 2428 Street 110 12th Street North, Suite F103  
 2429 City, State, Postal Code Birmingham, Alabama 35203  
 2430 Phone: +1 (205) 250-8053  
 2431 E-mail: dick@8760.com

2432  
 2433 Name David Burdett  
 2434 Company Commerce One  
 2435 Street 4400 Rosewood Drive  
 2436 City, State, Postal Code Pleasanton, CA 94588  
 2437 Country USA  
 2438 Phone: +1 (925) 520-4422  
 2439 EMail: david.burdett@commerceone.com

2440  
 2441 Name Chris Ferris  
 2442 Company Sun Microsystems  
 2443 Street One Network Drive  
 2444 City, State, Postal Code Burlington, MA 01803-0903  
 2445 Country USA  
 2446 Phone: +1 (781) 442-3063  
 2447 EMail: chris.ferris@east.sun.com

2448  
 2449 Name John Ibbotson  
 2450 Company IBM UK Ltd

2451 Street Hursley Park  
 2452 City, State, Postal Code Winchester SO21 2JN  
 2453 Country United Kingdom  
 2454 Phone: +44 (1962) 815188  
 2455 Email: john\_ibbotson@uk.ibm.com  
 2456  
 2457 Name Nicholas Kassem  
 2458 Company Java Software, Sun Microsystems  
 2459 Street 901 San Antonio Road, MS CUP02-201  
 2460 City, State, Postal Code Palo Alto, CA 94303-4900  
 2461 Phone: +1 (408) 863-3535  
 2462 E-mail: Nick.Kassem@eng.sun.com  
 2463  
 2464 Name Masayoshi Shimamura  
 2465 Company Fujitsu Limited  
 2466 Street Shinyokohama Nikko Bldg., 15-16, Shinyokohama 2-chome  
 2467 City, State, Postal Code Kohoku-ku, Yokohama 222-0033, Japan  
 2468 Phone: +81-45-476-4590  
 2469 E-mail: shima@rp.open.cs.fujitsu.co.jp  
 2470  
 2471 **Document Editing Team**  
 2472 Name Ralph Berwanger  
 2473 Company bTrade.com  
 2474 Street 2324 Gateway Drive  
 2475 City, State, Postal Code Irving, TX 75063  
 2476 Country USA  
 2477 Phone: +1 (972) 580-2900  
 2478 EMail: rberwanger@btrade.com  
 2479  
 2480 Name Ian Jones  
 2481 Company British Telecommunications  
 2482 Street Enterprise House, 84-85 Adam Street  
 2483 City, State, Postal Code Cardiff, CF24 2XF  
 2484 Country United Kingdom  
 2485 Phone: +44 29 2072 4063  
 2486 EMail: ian.c.jones@bt.com  
 2487  
 2488 Name Martha Warfelt  
 2489 Company Daimler Chrysler Corporation  
 2490 Street 800 Chrysler Drive  
 2491 City, State, Postal Code Auburn Hills, MI  
 2492 Country USA  
 2493 Phone: +1 (248) 944-5481 1210  
 2494 EMail: maw2@daimlerchrysler.com 1211

## 2495 Appendix A ebXMLHeader Schema and Data Type 2496 Definitions

### 2497 A.1 Schema Definition

2498 The following is the definition of the **ebXMLHeader** element as a schema that conforms to  
2499 [XMLSchema]. <DB>The few changes from version 0.91 are highlighted.</DB>

```

2500 <?xml version = "1.0" encoding = "UTF-8"?>
2501 <xsd:schema xmlns="http://www.ebxml.org/namespaces/messageHeader"
2502 targetNamespace="http://www.ebxml.org/namespaces/messageHeader"
2503 xmlns:ds="http://www.w3.org/2000/10/xmldsig#" xmlns:xlink="http://www.w3.org/1999/xlink"
2504 xmlns:xsd="http://www.w3.org/2000/10/XMLSchema">
2505   <xsd:import namespace="http://www.w3.org/2000/10/xmldsig#"
2506   schemaLocation="http://www.w3.org/TR/2000/10/xmldsig-core-schema/xmldsig-core-schema.xsd"/>
2507
2508 <!-- EBXML HEADER -->
2509 <xsd:element name="ebXMLHeader">
2510   <xsd:complexType>
2511     <xsd:sequence>
2512       <xsd:element ref="Manifest" minOccurs="0" maxOccurs="1"/>
2513       <xsd:element ref="Header"/>
2514       <xsd:element ref="RoutingHeaderList" minOccurs="0" maxOccurs="1"/>
2515       <xsd:element ref="Acknowledgment" minOccurs="0" maxOccurs="1"/>
2516       <xsd:element ref="StatusData" minOccurs="0" maxOccurs="1"/>
2517       <xsd:element ref="ApplicationHeaders" minOccurs="0" maxOccurs="1"/>
2518       <xsd:element ref="ErrorList" minOccurs="0" maxOccurs="1"/>
2519       <xsd:element ref="ds:Signature" minOccurs="0" maxOccurs="unbounded"/>
2520     </xsd:sequence>
2521     <xsd:attribute name="version" use="fixed" value="0.92" type="xsd:string"/>
2522     <xsd:anyAttribute namespace="##any" processContents="lax"/>
2523   </xsd:complexType>
2524 </xsd:element>
2525
2526 <!-- MANIFEST -->
2527 <xsd:element name="Manifest">
2528   <xsd:complexType>
2529     <xsd:sequence>
2530       <xsd:element ref="Reference" maxOccurs="unbounded"/>
2531       <xsd:any namespace="##other" processContents="lax"/>
2532     </xsd:sequence>
2533     <xsd:attribute name="id" use="required" type="xsd:ID"/>
2534   </xsd:complexType>
2535 </xsd:element>
2536
2537 <xsd:element name="Reference">
2538   <xsd:complexType>
2539     <xsd:sequence>
2540       <xsd:element ref="Schema" minOccurs="0" maxOccurs="unbounded"/>
2541       <xsd:element ref="Description" minOccurs="0" maxOccurs="1"/>
2542       <xsd:any namespace="##other" processContents="lax"/>
2543     </xsd:sequence>
2544     <xsd:attribute name="id" use="required" type="xsd:ID"/>
2545     <!-- Changed required to fixed on xlink:type -->
2546     <xsd:attribute name="xlink:type" use="fixed" type="xsd:string" value="simple"/>
2547     <xsd:attribute name="xlink:href" use="required" type="xsd:uriReference"/>
2548     <!-- Changed to optional on xlink:role -->
2549     <xsd:attribute name="xlink:role" type="xsd:uriReference"/>
2550   </xsd:complexType>
2551 </xsd:element>
2552
2553 <xsd:element name="Schema">
2554   <xsd:complexType>

```

```

2556 <xsd:simpleContent>
2557 <xsd:attribute name="location" use="required" type="xsd:uriReference"/>
2558 <xsd:attribute name="version" type="xsd:string"/>
2559 </xsd:simpleContent>
2560 </xsd:complexType>
2561 </xsd:element>
2562
2563 <!-- HEADER -->
2564 <xsd:element name="Header">
2565 <xsd:complexType>
2566 <xsd:sequence>
2567 <xsd:element ref="From"/>
2568 <xsd:element ref="To"/>
2569 <xsd:element ref="CPAId"/>
2570 <xsd:element ref="ConversationId"/>
2571 <xsd:element ref="Service"/>
2572 <xsd:element ref="Action"/>
2573 <xsd:element ref="MessageData"/>
2574 <!-- Changed Reliable Messaging Inf to Quality Of Service Info. -->
2575 <!-- Removed DeliveryReceiptRequested and TimeToLive and made them optional attributes of
2576 Quality of Service Info -->
2577 <xsd:element ref="QualityOfServiceInfo" minOccurs="0" maxOccurs="1"/>
2578 <!-- Changed description from maxOccurs 1 to unbounded -->
2579 <xsd:element ref="Description" minOccurs="0" maxOccurs="unbounded"/>
2580 <!-- Added SequenceNumber element -->
2581 <xsd:element ref="SequenceNumber" minOccurs="0" maxOccurs="1"/>
2582 <xsd:any namespace="##other" processContents="lax" minOccurs="0" maxOccurs="unbounded"/>
2583 </xsd:sequence>
2584 <xsd:attribute name="id" type="xsd:ID"/>
2585 </xsd:complexType>
2586 </xsd:element>
2587
2588 <xsd:element name="To">
2589 <xsd:complexType>
2590 <xsd:simpleContent>
2591 <xsd:extension base="xsd:string">
2592 <xsd:attribute name="type" type="xsd:string"/>
2593 </xsd:extension>
2594 </xsd:simpleContent>
2595 </xsd:complexType>
2596 </xsd:element>
2597
2598 <xsd:element name="CPAId" type="xsd:string"/>
2599
2600 <xsd:element name="ConversationId" type="xsd:string"/>
2601
2602 <xsd:element name="Service" type="xsd:string"/>
2603
2604 <xsd:element name="Action" type="xsd:string"/>
2605
2606 <xsd:element name="MessageData">
2607 <xsd:complexType>
2608 <xsd:sequence>
2609 <xsd:element ref="MessageId"/>
2610 <xsd:element ref="Timestamp"/>
2611 <xsd:element ref="RefToMessageId" minOccurs="0" maxOccurs="1"/>
2612 </xsd:sequence>
2613 </xsd:complexType>
2614 </xsd:element>
2615
2616 <xsd:element name="MessageId" type="xsd:string"/>
2617
2618 <xsd:element name="QualityOfServiceInfo">
2619 <xsd:complexType>
2620 <xsd:simpleContent>
2621 <xsd:attribute name="deliverySemantics" use="default" value="BestEffort"/>
2622 <xsd:simpleType>
2623 <xsd:restriction base="xsd:NMTOKEN">
2624 <xsd:enumeration value="OnceAndOnlyOnce"/>
2625 <xsd:enumeration value="BestEffort"/>
2626 </xsd:restriction>

```

```

2627     </xsd:simpleType>
2628 <!-- Added in messageOrderSemantics attribute -->
2629     <xsd:attribute name="messageOrderSemantics" use="default" value="NotGuaranteed"/>
2630     <xsd:simpleType>
2631         <xsd:restriction base="xsd:NMTOKEN">
2632             <xsd:enumeration value="Guaranteed"/>
2633             <xsd:enumeration value="NotGuaranteed"/>
2634         </xsd:restriction>
2635     </xsd:simpleType>
2636 <!-- Added in deliveryReceiptRequested attribute -->
2637     <xsd:attribute name="deliveryReceiptRequested" use="default" value="None"/>
2638     <xsd:simpleType>
2639         <xsd:restriction base="xsd:NMTOKEN">
2640             <xsd:enumeration value="Signed"/>
2641             <xsd:enumeration value="UnSigned"/>
2642             <xsd:enumeration value="None"/>
2643         </xsd:restriction>
2644     </xsd:simpleType>
2645 <!-- Added in timeToLive attribute -->
2646     <xsd:attribute name="timeToLive" type="xsd:timeInstant"/>
2647 </xsd:simpleContent>
2648 </xsd:complexType>
2649 </xsd:element>
2650
2651 <!-- ROUTING HEADER LIST -->
2652 <xsd:element name="RoutingHeaderList">
2653     <xsd:complexType>
2654         <xsd:sequence>
2655             <xsd:element ref="RoutingHeader" maxOccurs="unbounded"/>
2656         </xsd:sequence>
2657         <xsd:attribute name="id" type="xsd:ID"/>
2658     </xsd:complexType>
2659 </xsd:element>
2660
2661 <xsd:element name="RoutingHeader">
2662     <xsd:complexType>
2663         <xsd:sequence>
2664             <xsd:element ref="SenderURI"/>
2665             <xsd:element ref="ReceiverURI"/>
2666             <xsd:element ref="ErrorURI" minOccurs="0" maxOccurs="1"/>
2667             <xsd:element ref="Timestamp"/>
2668             <xsd:element ref="SequenceNumber" minOccurs="0" maxOccurs="1"/>
2669             <xsd:any namespace="##other" processContents="lax" minOccurs="0" maxOccurs="unbounded"/>
2670         </xsd:sequence>
2671         <xsd:attribute name="reliableMessagingMethod"/>
2672     <xsd:simpleType>
2673         <xsd:restriction base="xsd:NMTOKEN">
2674             <xsd:enumeration value="ebXML"/>
2675             <xsd:enumeration value="Transport"/>
2676         </xsd:restriction>
2677     </xsd:simpleType>
2678     <xsd:attribute name="intermediateAckRequested"/>
2679     <xsd:simpleType>
2680         <xsd:restriction base="xsd:NMTOKEN">
2681             <xsd:enumeration value="Signed"/>
2682             <xsd:enumeration value="UnSigned"/>
2683             <xsd:enumeration value="None"/>
2684         </xsd:restriction>
2685     </xsd:simpleType>
2686 </xsd:complexType>
2687 </xsd:element>
2688
2689 <xsd:element name="SenderURI" type="xsd:uriReference"/>
2690
2691 <xsd:element name="ReceiverURI" type="xsd:uriReference"/>
2692
2693 <xsd:element name="SequenceNumber" type="xsd:positiveInteger" minOccurs="0" maxOccurs="1"/>
2694
2695 <xsd:element name="ErrorURI" type="xsd:uriReference" minOccurs="0" maxOccurs="1"/>
2696
2697 <!-- APPLICATION HEADERS -->

```

```

2698 <xsd:element name="ApplicationHeaders" type="ApplicationHeaders"/>
2699 <xsd:complexType name="ApplicationHeaders">
2700 <xsd:sequence>
2701 <xsd:any namespace="##other" processContents="lax"/>
2702 </xsd:sequence>
2703 <xsd:attribute name="id" type="xsd:ID"/>
2704 </xsd:complexType>
2705
2706 <!-- ACKNOWLEDGEMENT -->
2707 <xsd:element name="Acknowledgment">
2708 <xsd:complexType>
2709 <xsd:sequence>
2710 <xsd:element ref="Timestamp"/>
2711 <xsd:element ref="From" minOccurs="0" maxOccurs="1"/>
2712 </xsd:sequence>
2713 <xsd:attribute name="id" type="xsd:ID"/>
2714 <xsd:attribute name="type" use="default" value="DeliveryReceipt"/>
2715 <xsd:simpleType>
2716 <xsd:restriction base="xsd:NMTOKEN">
2717 <xsd:enumeration value="DeliveryReceipt"/>
2718 <xsd:enumeration value="IntermediateAck"/>
2719 </xsd:restriction>
2720 </xsd:simpleType>
2721 <xsd:attribute name="signed" type="xsd:boolean"/>
2722 </xsd:complexType>
2723 </xsd:element>
2724
2725 <!-- ERROR LIST -->
2726 <xsd:element name="ErrorList">
2727 <xsd:complexType>
2728 <xsd:sequence>
2729 <xsd:element ref="Error" maxOccurs="unbounded"/>
2730 </xsd:sequence>
2731 <xsd:attribute name="id" type="xsd:ID"/>
2732 <xsd:attribute name="highestSeverity" use="default" value="Warning"/>
2733 <xsd:simpleType>
2734 <xsd:restriction base="xsd:string">
2735 <xsd:enumeration value="Warning"/>
2736 <xsd:enumeration value="Error"/>
2737 </xsd:restriction>
2738 </xsd:simpleType>
2739 </xsd:complexType>
2740 </xsd:element>
2741
2742 <xsd:element name="Error">
2743 <xsd:complexType>
2744 <xsd:attribute name="codeContext" use="required" type="xsd:uriReference"/>
2745 <xsd:attribute name="errorCode" use="required" type="xsd:string"/>
2746 <xsd:attribute name="severity" use="default" value="Warning"/>
2747 <xsd:simpleType>
2748 <xsd:restriction base="xsd:NMTOKEN">
2749 <xsd:enumeration value="Warning"/>
2750 <xsd:enumeration value="Error"/>
2751 </xsd:restriction>
2752 </xsd:simpleType>
2753 <xsd:attribute name="location" type="xsd:string"/>
2754 <xsd:attribute name="xml:lang" type="xsd:language"/>
2755 <xsd:attribute name="errorMessage" type="xsd:string"/>
2756 <xsd:attribute name="softwareDetails" type="xsd:string"/>
2757 </xsd:complexType>
2758 </xsd:element>
2759
2760 <!-- STATUS DATA -->
2761 <xsd:element name="StatusData">
2762 <xsd:sequence>
2763 <xsd:element ref="RefToMessageId"/>
2764 <xsd:element ref="Timestamp" minOccurs="0" maxOccurs="1"/>
2765 <xsd:element name="ForwardURI" type="xsd:uriReference" minOccurs="0" maxOccurs="1"/>
2766 </xsd:sequence>
2767 <xsd:attribute name="messageStatus"/>
2768 <xsd:simpleType>

```

```

2769     <xsd:restriction base="xsd:NMTOKEN">
2770     <xsd:enumeration value="Unauthorized"/>
2771     <xsd:enumeration value="NotRecognized"/>
2772     <xsd:enumeration value="Received"/>
2773     <xsd:enumeration value="Processed"/>
2774     <xsd:enumeration value="Forwarded"/>
2775     </xsd:restriction>
2776 </xsd:simpleType>
2777 </xsd:element>
2778
2779 <!-- COMMON ELEMENTS -->
2780 <xsd:element name="From">
2781   <xsd:complexType>
2782     <xsd:simpleContent>
2783       <xsd:extension base="xsd:string">
2784         <xsd:attribute name="type" type="xsd:string"/>
2785       </xsd:extension>
2786     </xsd:simpleContent>
2787   </xsd:complexType>
2788 </xsd:element>
2789
2790 <xsd:element name="Description">
2791   <xsd:complexType>
2792     <xsd:simpleContent>
2793       <xsd:extension base="xsd:string">
2794         <xsd:attribute name="xml:lang" type="xsd:NMTOKEN"/>
2795       </xsd:extension>
2796     </xsd:simpleContent>
2797   </xsd:complexType>
2798 </xsd:element>
2799
2800 <xsd:element name="RefToMessageId" type="xsd:string"/>
2801
2802 <xsd:element name="Timestamp" type="xsd:timeInstant"/>
2803 <!-- Does timeInstant conform to ISO 2601? -->
2804
2805 </xsd:schema>
2806 <?xml version="1.0" encoding="UTF-8"?>
2807 <xsd:schema xmlns="http://www.ebxml.org/namespaces/messageHeader"
2808   targetNamespace="http://www.ebxml.org/namespaces/messageHeader"
2809   xmlns:ds="http://www.w3.org/2000/10/xmlsig#" xmlns:xlink="http://www.w3.org/1999/xlink"
2810   xmlns:xsd="http://www.w3.org/2000/10/XMLSchema">
2811   <xsd:import namespace="http://www.w3.org/2000/10/xmlsig#"
2812     schemaLocation="http://www.w3.org/TR/2000/10/xmlsig-core-schema/xmlsig-core-schema.xsd"/>
2813
2814   <!-- EBXML HEADER -->
2815   <xsd:element name="ebXMLHeader">
2816     <xsd:complexType>
2817       <xsd:sequence>
2818         <xsd:element ref="Manifest" minOccurs="0" maxOccurs="1"/>
2819         <xsd:element ref="Header"/>
2820         <xsd:element ref="RoutingHeaderList" minOccurs="0" maxOccurs="1"/>
2821         <xsd:element ref="Acknowledgment" minOccurs="0" maxOccurs="1"/>
2822         <xsd:element ref="StatusData" minOccurs="0" maxOccurs="1"/>
2823         <xsd:element ref="ApplicationHeaders" minOccurs="0" maxOccurs="1"/>
2824         <xsd:element ref="ErrorList" minOccurs="0" maxOccurs="1"/>
2825         <xsd:element ref="ds:Signature" minOccurs="0" maxOccurs="unbounded"/>
2826       </xsd:sequence>
2827       <xsd:attribute name="version" use="fixed" value="0.9" type="xsd:string"/>
2828       <xsd:anyAttribute namespace="##any" processContents="lax"/>
2829     </xsd:complexType>
2830   </xsd:element>
2831
2832   <!-- MANIFEST -->
2833   <xsd:element name="Manifest">
2834     <xsd:complexType>
2835       <xsd:sequence>
2836         <xsd:element ref="Reference" maxOccurs="unbounded"/>
2837         <xsd:any namespace="##other" processContents="lax"/>
2838       </xsd:sequence>
2839     <xsd:attribute name="id" use="required" type="xsd:ID"/>

```

```

2840 </xsd:complexType>
2841 </xsd:element>
2842
2843 <xsd:element name="Reference">
2844 <xsd:complexType>
2845 <xsd:sequence>
2846 <xsd:element ref="Schema" minOccurs="0" maxOccurs="1"/>
2847 <xsd:element ref="Description" minOccurs="0" maxOccurs="1"/>
2848 <xsd:any namespace="##other" processContents="lax"/>
2849 </xsd:sequence>
2850 <xsd:attribute name="id" type="xsd:ID"/>
2851 <xsd:attribute name="xlink:type" use="required" type="xsd:string" value="simple"/>
2852 <xsd:attribute name="xlink:href" use="required" type="xsd:uriReference"/>
2853 <xsd:attribute name="xlink:label" type="xsd:string"/>
2854 <xsd:attribute name="xlink:role" use="required" type="xsd:uriReference"/>
2855 <xsd:attribute name="xlink:title" type="xsd:string"/>
2856 </xsd:complexType>
2857 </xsd:element>
2858
2859 <xsd:element name="Schema">
2860 <xsd:complexType>
2861 <xsd:simpleContent>
2862 <xsd:attribute name="location" use="required" type="xsd:string"/>
2863 <xsd:attribute name="version" use="required" type="xsd:string"/>
2864 </xsd:simpleContent>
2865 </xsd:complexType>
2866 </xsd:element>
2867
2868 <!-- HEADER -->
2869 <xsd:element name="Header">
2870 <xsd:complexType>
2871 <xsd:sequence>
2872 <xsd:element ref="From"/>
2873 <xsd:element ref="To"/>
2874 <xsd:element ref="CPAId"/>
2875 <xsd:element ref="ConversationId"/>
2876 <xsd:element ref="Service"/>
2877 <xsd:element ref="Action"/>
2878 <xsd:element ref="MessageData"/>
2879 <!-- Removed DeliveryReceiptRequested and TimeToLive and made them optional attributes of
2880 Reliable Messaging Info -->
2881 <xsd:element ref="ReliableMessagingInfoQualityOfServiceInfo" minOccurs="0"
2882 maxOccurs="1"/>
2883 <xsd:element ref="Description" minOccurs="0" maxOccurs="1"/>
2884 <xsd:any namespace="##other" processContents="lax" minOccurs="0" maxOccurs="unbounded"/>
2885 </xsd:sequence>
2886 <xsd:attribute name="id" type="xsd:ID"/>
2887 </xsd:complexType>
2888 </xsd:element>
2889
2890 <xsd:element name="To">
2891 <xsd:complexType>
2892 <xsd:simpleContent>
2893 <xsd:extension base="xsd:string">
2894 <xsd:attribute name="type" type="xsd:string"/>
2895 </xsd:extension>
2896 </xsd:simpleContent>
2897 </xsd:complexType>
2898 </xsd:element>
2899
2900 <xsd:element name="CPAId" type="xsd:string"/>
2901
2902 <xsd:element name="ConversationId" type="xsd:string"/>
2903
2904 <xsd:element name="Service" type="xsd:string"/>
2905
2906 <xsd:element name="Action" type="xsd:string"/>
2907
2908 <xsd:element name="MessageData">
2909 <xsd:complexType>
2910 <xsd:sequence>

```

```

2911 <!-->
2912 <!-->
2913 <!-->
2914 <!-->
2915 <!-->
2916 <!-->
2917 <!-->
2918 <!-->
2919 <!-->
2920 <!-->
2921 <!-->
2922 <!-->
2923 <!-->
2924 <!-->
2925 <!-->
2926 <!-->
2927 <!-->
2928 <!-->
2929 <!-->
2930 <!-- Added in deliveryReceiptRequested attribute -->
2931 <!-->
2932 <!-->
2933 <!-->
2934 <!-->
2935 <!-->
2936 <!-->
2937 <!-->
2938 <!-->
2939 <!-- Added in timeToLive attribute -->
2940 <!-->
2941 <!-->
2942 <!-->
2943 <!-->
2944 <!-->
2945 <!-- ROUTING HEADER LIST -->
2946 <!-->
2947 <!-->
2948 <!-->
2949 <!-->
2950 <!-->
2951 <!-->
2952 <!-->
2953 <!-->
2954 <!-->
2955 <!-->
2956 <!-->
2957 <!-->
2958 <!-->
2959 <!-->
2960 <!-->
2961 <!-->
2962 <!-->
2963 <!-->
2964 <!-->
2965 <!-->
2966 <!-->
2967 <!-->
2968 <!-->
2969 <!-->
2970 <!-->
2971 <!-->
2972 <!-->
2973 <!-->
2974 <!-->
2975 <!-->
2976 <!-->
2977 <!-->
2978 <!-->
2979 <!-->
2980 <!-->
2981 <!-->

```

```

2982
2983 -----<xsd:element name="SenderURI" type="xsd:uriReference"/>
2984
2985 -----<xsd:element name="ReceiverURI" type="xsd:uriReference"/>
2986
2987 -----<xsd:element name="SequenceNumber" type="xsd:positiveInteger" minOccurs="0" maxOccurs="1"/>
2988
2989 -----<xsd:element name="ErrorURI" type="xsd:uriReference" minOccurs="0" maxOccurs="1"/>
2990
2991 <!-- APPLICATION HEADERS -->
2992 -----<xsd:element name="ApplicationHeaders" type="ApplicationHeaders"/>
2993 -----<xsd:complexType name="ApplicationHeaders">
2994 -----<xsd:sequence>
2995 -----<xsd:any namespace="##other" processContents="lax"/>
2996 -----</xsd:sequence>
2997 -----<xsd:attribute name="id" type="xsd:ID"/>
2998 -----</xsd:complexType>
2999
3000 <!-- ACKNOWLEDGEMENT/ACKNOWLEDGMENT -->
3001 -----<xsd:element name="Acknowledgment">
3002 -----<xsd:complexType>
3003 -----<xsd:sequence>
3004 -----<xsd:element ref="Timestamp"/>
3005 -----<xsd:element ref="From" minOccurs="0" maxOccurs="1"/>
3006 -----</xsd:sequence>
3007 -----<xsd:attribute name="id" type="xsd:ID"/>
3008 -----<xsd:attribute name="type" use="default" value="DeliveryReceipt"/>
3009 -----<xsd:simpleType>
3010 -----<xsd:restriction base="xsd:NMTOKEN">
3011 -----<xsd:enumeration value="DeliveryReceipt"/>
3012 -----<xsd:enumeration value="IntermediateAck"/>
3013 -----</xsd:restriction>
3014 -----</xsd:simpleType>
3015 -----<xsd:attribute name="signed" type="xsd:boolean"/>
3016 -----</xsd:complexType>
3017 -----</xsd:element>
3018
3019 <!-- ERROR LIST -->
3020 -----<xsd:element name="ErrorList">
3021 -----<xsd:complexType>
3022 -----<xsd:sequence>
3023 -----<xsd:element ref="Error" maxOccurs="unbounded"/>
3024 -----</xsd:sequence>
3025 -----<xsd:attribute name="id" type="xsd:ID"/>
3026 -----<xsd:attribute name="highestSeverity" use="default" value="Warning"/>
3027 -----<xsd:simpleType>
3028 -----<xsd:restriction base="xsd:string">
3029 -----<xsd:enumeration value="Warning"/>
3030 -----<xsd:enumeration value="Error"/>
3031 -----</xsd:restriction>
3032 -----</xsd:simpleType>
3033 -----</xsd:complexType>
3034 -----</xsd:element>
3035
3036 -----<xsd:element name="Error">
3037 -----<xsd:complexType>
3038 -----<xsd:attribute name="codeContext" use="required" type="xsd:uriReference"/>
3039 -----<xsd:attribute name="errorCode" use="required" type="xsd:string"/>
3040 -----<xsd:attribute name="severity" use="default" value="Warning"/>
3041 -----<xsd:simpleType>
3042 -----<xsd:restriction base="xsd:NMTOKEN">
3043 -----<xsd:enumeration value="Warning"/>
3044 -----<xsd:enumeration value="Error"/>
3045 -----</xsd:restriction>
3046 -----</xsd:simpleType>
3047 -----<xsd:attribute name="location" type="xsd:string"/>
3048 -----<xsd:attribute name="xml:lang" type="xsd:language"/>
3049 -----<xsd:attribute name="errorMessage" type="xsd:string"/>
3050 -----<xsd:attribute name="softwareDetails" type="xsd:string"/>
3051 -----</xsd:complexType>
3052 -----</xsd:element>

```

```

3053
3054 <!-- STATUS DATA -->
3055 <xsd:element name="StatusData">
3056 <xsd:sequence>
3057 <xsd:element ref="RefToMessageId"/>
3058 <xsd:element ref="Timestamp" minOccurs="0" maxOccurs="1"/>
3059 <xsd:element name="ForwardURI" type="xsd:uriReference" minOccurs="0" maxOccurs="1"/>
3060 </xsd:sequence>
3061 <xsd:attribute name="messageStatus"/>
3062 <xsd:simpleType>
3063 <xsd:restriction base="xsd:NMTOKEN">
3064 <xsd:enumeration value="Unauthorized"/>
3065 <xsd:enumeration value="NotRecognized"/>
3066 <xsd:enumeration value="Received"/>
3067 <xsd:enumeration value="Processed"/>
3068 <xsd:enumeration value="Forwarded"/>
3069 </xsd:restriction>
3070 </xsd:simpleType>
3071 </xsd:element>
3072
3073 <!-- COMMON ELEMENTS -->
3074 <xsd:element name="From">
3075 <xsd:complexType>
3076 <xsd:simpleContent>
3077 <xsd:extension base="xsd:string">
3078 <xsd:attribute name="type" type="xsd:string"/>
3079 </xsd:extension>
3080 </xsd:simpleContent>
3081 </xsd:complexType>
3082 </xsd:element>
3083
3084 <xsd:element name="Description">
3085 <xsd:complexType>
3086 <xsd:simpleContent>
3087 <xsd:extension base="xsd:string">
3088 <xsd:attribute name="xml:lang" type="xsd:NMTOKEN"/>
3089 </xsd:extension>
3090 </xsd:simpleContent>
3091 </xsd:complexType>
3092 </xsd:element>
3093
3094 <xsd:element name="RefToMessageId" type="xsd:string"/>
3095
3096 <xsd:element name="Timestamp" type="xsd:timeInstant"/>
3097 <!-- Does timeInstant conform to ISO-2601? -->
3098
3099 </xsd:schema>

```

## 3100 A.2 Data Type Definition

3101 This section will contain a [XML] DTD that is equivalent to the schema defined in section A.1.

3102 **Appendix B Examples**

3103 To be completed.

3104 **Appendix C Communication Protocol Interfaces**

3105 This Appendix describes how the ebXML Message Service messages are carried by  
 3106 Communication Protocols. Two protocols are supported:

- 3107 • Hypertext Transfer Protocol – HTTP/1.1, in both asynchronous and synchronous forms,  
 3108 and
- 3109 • SMTP – Simple Mail Transfer Protocol

3110 **C.1 HTTP**

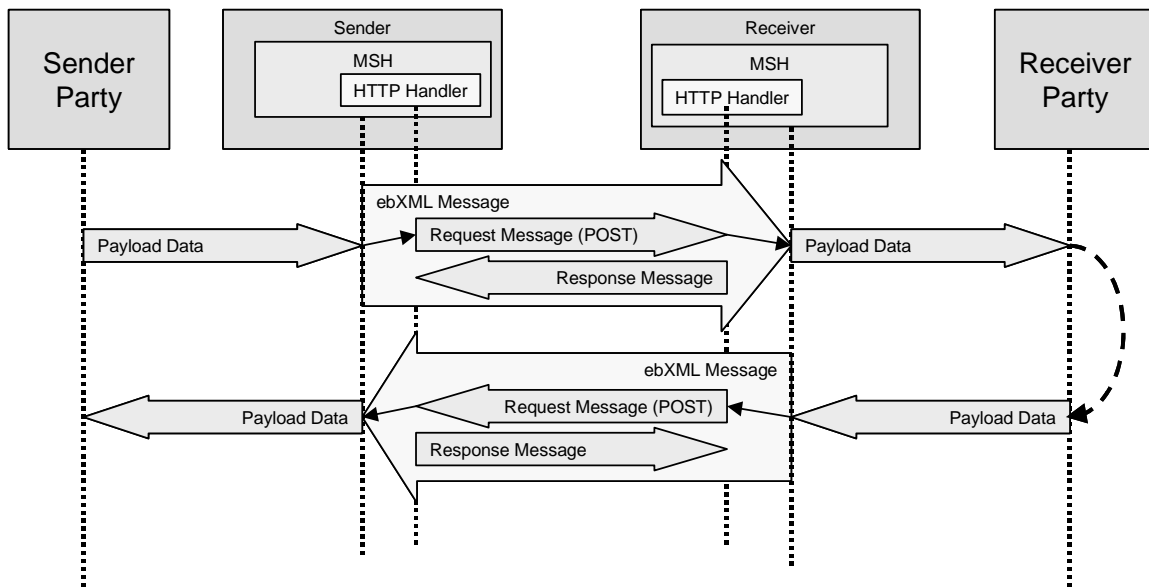
3111 This section describes how to transport ebXML compliant messages of [HTTP]. This can work in  
 3112 one of the following two ways:

- 3113 • asynchronously, where the response to a message is sent using a separate HTTP POST,  
 3114 and
- 3115 • synchronously, where the response to a message is sent on the HTTP RESPONSE  
 3116 returned from an HTTP POST

3117 These are described below.

3118 **C.1.1 Asynchronous HTTP**

3119 In Asynchronous HTTP, all ebXML Message Service messages are carried by an HTTP Request  
 3120 Message (POST method). The HTTP Response Message to an HTTP Request Message has no  
 3121 entity body. This is illustrated by the figure below.



3122  
 3123 **Figure C.1 Asynchronous HTTP Message Flow**

3124 A message that is being sent asynchronously MAY be identified by the following HTTP header:

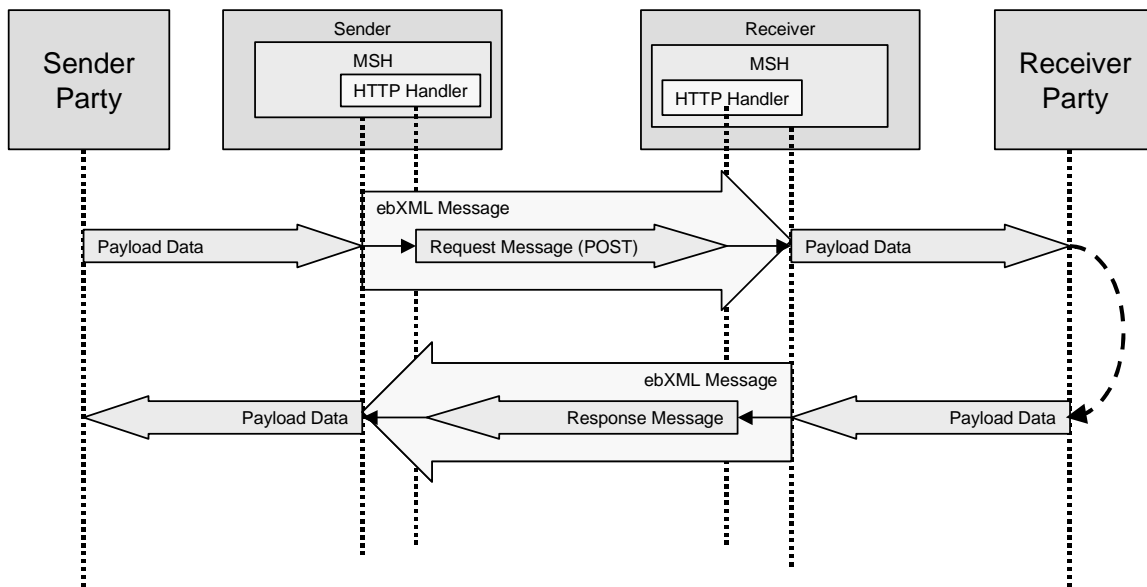
3125 `ebxmlresponse=asynchronous`

3126 If the `ebXMLresponse` HTTP parameter is omitted then it MUST be assumed that the response  
 3127 is sent asynchronously.

3128 **C.1.2 Synchronous HTTP**

3129 *[The Synchronous HTTP section has not been agreed to by the membership of the TRP*  
 3130 *Project Team; however, it is being included to provide a basis for POC developers of MSH*  
 3131 *implementations. Implementers MUST be prepared for some change to the content of this*  
 3132 *section.]*

3133 In Synchronous HTTP, one ebXML Message Service message is carried by an HTTP Request  
 3134 Message (POST method) with the ebXML Message that is a response to the first message sent  
 3135 in the HTTP Response Message to the HTTP Request Message. This is illustrated by the figure  
 3136 below.



3137

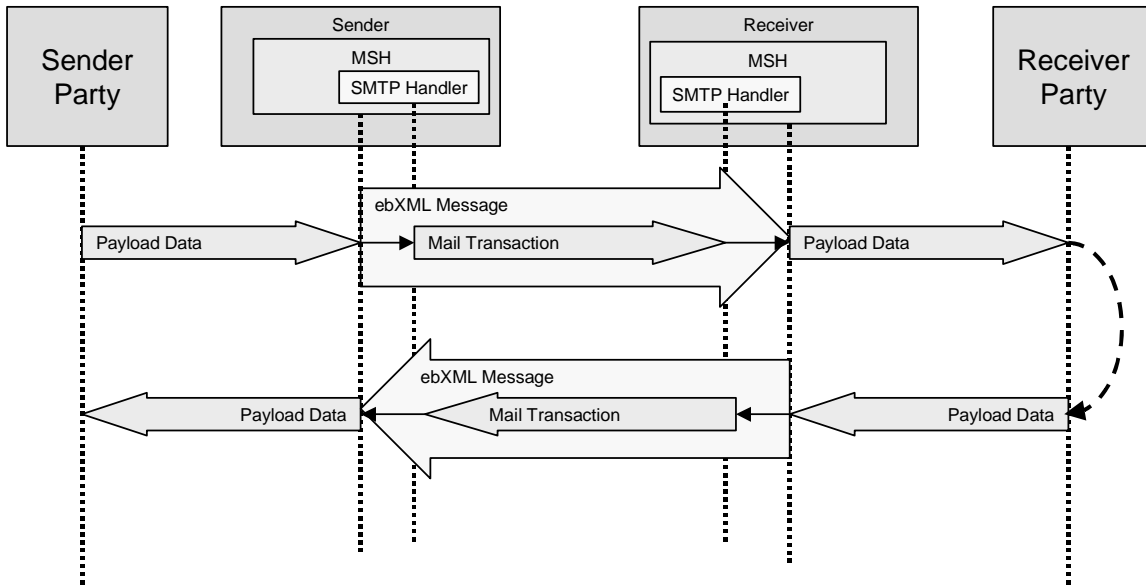
3138 **Figure C.2 Synchronous HTTP Message Flow**

3139 If a response is being sent synchronously, the following HTTP header MUST be included in the  
 3140 HTTP envelope:

3141 `ebxmlresponse=synchronous`

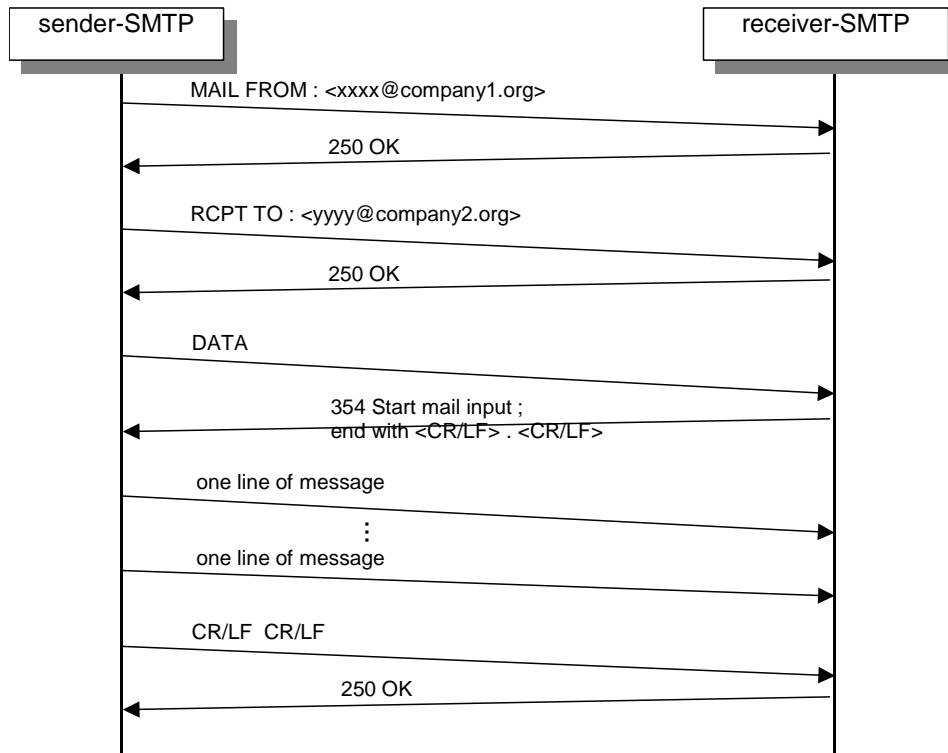
3142 **C.2 SMTP**

3143 All ebXML Message Service messages are carried as mail in an [SMTP] Mail Transaction as  
 3144 shown in the figure below.



3145  
 3146 **Figure C.3 SMTP Message Flow**

3147 The Mail Transaction follows RFC 821, "SIMPLE MAIL TRANSFER PROTOCOL", as shown in  
 3148 the following Figure:



3149  
 3150 **Figure C.4 SMTP Sequence**

**3151 C.3 FTP**

3152 This section will describe how ebXML Messages may be sent using the File Transfer protocol as  
3153 defined in RFC 959

3154 This section is to be completed.

**3155 C.4 Communication Protocol Errors****3156 C.4.1 Use of Error Codes**

3157 Communication Protocol Error Codes are used only to report errors in the communication  
3158 protocol envelope (see section 7.1). A normal OK Response (e.g. an HTTP code 200) is used  
3159 even if there are errors in the MIME envelope, the ebXML Header document or the payload.

**3160 C.4.2 Communication Errors during Reliable Messaging**

3161 When the Sender or the Receiver detects a transport protocol level error (such as an HTTP,  
3162 SMTP or FTP error) and Reliable Messaging is being used then the appropriate transport  
3163 recovery handler will execute a recovery sequence. Only if the error is unrecoverable, does  
3164 Reliable Messaging recovery take place (see section 10).

3165 **Appendix D Reliable Messaging Processing Logic**

3166 This section will contain non-normative reference processing logic to describe the behavior of a  
3167 MSH that is taking part in reliable messaging. It's purpose is to assist implementers in developing  
3168 consistent interoperable solutions.

## 3169 **Copyright Statement**

3170 This document and translations of it may be copied and furnished to others, and derivative works  
3171 that comment on or otherwise explain it or assist in its implementation may be prepared, copied,  
3172 published and distributed, in whole or in part, without restriction of any kind, provided that the  
3173 above copyright notice and this paragraph are included on all such copies and derivative works.  
3174 However, this document itself may not be modified in any way, such as by removing the copyright  
3175 notice or references to the Internet Society or other Internet organizations, except as needed for  
3176 the purpose of developing Internet standards in which case the procedures for copyrights defined  
3177 in the Internet Standards process must be followed, or as required to translate it into languages  
3178 other than English.

3179 The limited permissions granted above are perpetual and will not be revoked by ebXML or its  
3180 successors or assigns.

3181 This document and the information contained herein is provided on an "AS IS" basis and ebXML  
3182 DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO  
3183 ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE  
3184 ANY RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A  
3185 PARTICULAR PURPOSE.