



Creating A Single Global Electronic Market

Message Service Specification

ebXML Transport, Routing & Packaging

Version 0.93~~0.92~~

~~2-18 February 2001~~ 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

Latest version

<http://www.ebxml.org>

Previous version

<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 – ~~Precess~~ Progress Software
Martha Warfelt – Daimler Chrysler
Prasad Yendluri – Web Methods

3 Table of Contents

- 1 Status of this Document 2
- 2 ebXML Participants 3
- 3 Table of Contents 4
- 4 Introduction 8
 - 4.1 Summary of Contents of Document 8
 - 4.2 Document Conventions 9
 - 4.3 Audience 9
 - 4.4 Caveats and Assumptions 9
 - 4.5 Related Documents 9
- 5 Design Objectives 11
- 6 System Overview 12
 - 6.1 What the Message Service does 12
 - 6.2 Message Service Overview 12
- 7 Packaging Specification 14
 - 7.1 Introduction 14
 - 7.1.1 ebXML Header Envelope and ebXML Payload Envelope 14
 - 7.2 ebXML Message Envelope 15
 - 7.2.1 Content-Type 15
 - 7.2.1.1 type Attribute 15
 - 7.2.1.2 boundary Attribute 15
 - 7.2.1.3 version Attribute 15
 - 7.2.2 ebXML Message Envelope Example 15
 - 7.3 ebXML Header Container 16
 - 7.3.1 Content-ID 16
 - 7.3.2 Content-Type 16
 - 7.3.2.1 version Attribute 16
 - 7.3.2.2 charset Attribute 16
 - 7.3.3 ebXML Header Envelope Example 17
 - 7.4 ebXML Payload Container 17
 - 7.4.1 Content-ID 18
 - 7.4.2 Content-Type 18
 - 7.4.3 Example of an ebXML MIME Payload Container 18
 - 7.5 Additional MIME Parameters 18
 - 7.6 Reporting MIME Errors 18
- 8 ebXML Header Document 19
 - 8.1 XML Prolog 19
 - 8.1.1 XML Declaration 19
 - 8.1.2 Encoding Declaration 19
 - 8.1.3 Standalone Document Declaration 19
 - 8.1.4 Document Type Declaration 20
 - 8.2 ebXMLHeader Element 20
 - 8.2.1 ebXMLHeader attributes 20
 - 8.2.1.1 Namespace attribute 20
 - 8.2.1.2 version attribute 20
 - 8.2.2 ebXMLHeader elements 20

- 8.2.3 Combining Principal Header Elements 21
 - 8.2.3.1 Manifest element 21
 - 8.2.3.2 Header element 21
 - 8.2.3.3 RoutingHeaderList element 21
 - 8.2.3.4 ApplicationHeaders element 21
 - 8.2.3.5 StatusData element 21
 - 8.2.3.6 ErrorList element 21
 - 8.2.3.7 Acknowledgment element 21
 - 8.2.3.8 Signature element 21
 - 8.2.3.9 #wildcard element content 22
- 8.2.4 ebXMLHeader sample 22
- 8.3 Manifest element 22
 - 8.3.1 Reference element 22
 - 8.3.1.1 Schema element 23
 - 8.3.1.2 Description element 23
 - 8.3.1.3 #wildcard element 23
 - 8.3.2 What References are Included in a Manifest 23
 - 8.3.3 Manifest Validation 23
 - 8.3.4 Manifest sample 23
- 8.4 Header element 24
 - 8.4.1 From and To elements 24
 - 8.4.2 CPAlid element 24
 - 8.4.3 ConversationId element 25
 - 8.4.4 Service element 25
 - 8.4.4.1 type attribute 25
 - 8.4.4.2 ebXML Message Service namespace 25
 - 8.4.5 Action element 25
 - 8.4.6 MessageData element 26
 - 8.4.6.1 MessageId element 26
 - 8.4.6.2 Timestamp element 26
 - 8.4.6.3 RefToMessageId element 26
 - 8.4.6.4 TimeToLive element 26
 - 8.4.7 QualityOfServiceInfo element 27
 - 8.4.7.1 deliverySemantics attribute 27
 - 8.4.7.1 messageOrderSemantics attribute 27
 - 8.4.7.2 DeliveryReceiptRequested attribute 28
 - 8.4.7.3 syncReplyMode attribute 28
 - 8.4.8 SequenceNumber element 29
 - 8.4.9 Description element 30
 - 8.4.10 #wildcard element 30
 - 8.4.11 Header sample 30
- 8.5 RoutingHeaderList element 30
 - 8.5.1 Routing Header Element 31
 - 8.5.1.1 reliableMessagingMethod attribute 31
 - 8.5.1.2 intermediateAckRequested attribute 31
 - 8.5.1.3 SenderURI element 31
 - 8.5.1.4 ReceiverURI element 32
 - 8.5.1.5 ErrorURI element 32
 - 8.5.1.6 Timestamp element 32
 - 8.5.1.7 SequenceNumber element 32
 - 8.5.1.8 #wildcard element 32
 - 8.5.2 Single Hop Routing Header Sample 33
 - 8.5.3 Multi-hop Routing Header Sample 34
- 8.6 ApplicationHeaders Element 35
 - 8.6.1 ApplicationHeaders sample 35
- 8.7 StatusData Element 35
- 8.8 ErrorList Element 36
 - 8.8.1 id attribute 36
 - 8.8.2 highestSeverity attribute 36
 - 8.8.3 Error element 36
 - 8.8.3.1 codeContext attribute 36

8.8.3.2	errorCode attribute	36
8.8.3.3	severity attribute	36
8.8.3.4	location attribute	37
8.8.3.5	errorMessage attribute	37
8.8.3.6	softwareDetails attribute	37
8.8.4	Examples	37
8.8.5	errorCode values	38
8.8.6	Reporting Errors in the ebXML Header Document	38
8.8.7	Non-XML Document Errors	38
8.9	Acknowledgment Element	39
8.9.1	Timestamp element	40
8.9.2	From element	40
8.9.3	type attribute	40
8.9.4	signed attribute	40
8.10	Signature Element	40
9	Message Service Handler Services	41
9.1	Message Status Request Service	41
9.1.1	Message Status Request Message	41
9.1.2	Message Status Response Message	41
9.1.3	Security Considerations	42
9.2	Message Service Handler Ping Service	42
9.2.1	Message Service Handler Ping Message	42
9.2.2	Message Service Handler Pong Message	42
9.2.3	Security Considerations	43
10	Reliable Messaging	44
10.1.1	Persistent Storage and System Failure	44
10.1.2	Methods of Implementing Reliable Messaging	44
10.2	ebXML Reliable Messaging Protocol	44
10.2.1	Single-hop Reliable Messaging	45
10.2.1.1	Sending Message Behavior	45
10.2.1.2	Receiving Message Behavior	46
10.2.1.3	Resending Lost Messages and Duplicate Filtering	47
10.2.2	Multi-hop Reliable Messaging	49
10.2.2.1	Multi-hop Reliable Messaging without Intermediate Acknowledgments	49
10.2.2.2	Multi-hop Reliable Messaging with Intermediate Acknowledgments	50
10.3	ebXML Reliable Messaging using Queuing Transports	51
10.4	Service and Action Element Values	52
10.5	Failed Message Delivery	52
10.6	Reliable Messaging Parameters	53
10.6.1	Who sets Message Service Parameters	53
10.6.2	From Party Parameters	54
10.6.2.1	Delivery Semantics	54
10.6.2.2	Delivery Receipt Requested	55
10.6.2.3	Sync Reply Mode	55
10.6.2.4	Time To Live	55
10.6.3	To Party Parameters	55
10.6.3.1	Delivery Receipt Provided	55
10.6.4	Sending MSH Parameters	55
10.6.4.1	Reliable Messaging Method	55
10.6.4.2	Intermediate Ack Requested	55
10.6.4.3	Timeout Parameter	56
10.6.4.4	Retries Parameter	56
10.6.4.5	RetryInterval Parameter	56
10.6.4.6	Deciding when to resend a message	56
10.6.5	Receiving MSH Parameters	56
10.6.5.1	Reliable Messaging Methods Supported	56
10.6.5.2	PersistDuration	56

10.6.5.3 MSH Time Accuracy..... 57

11 Error Reporting and Handling 58

11.1 Definitions 58

11.2 Types of Errors 58

11.3 When to generate Error Messages 58

11.3.1 Security Considerations..... 58

11.4 Identifying the Error Reporting Location..... 58

11.5 Service and Action Element Values 59

12 Security 60

12.1 Security and Management..... 60

12.2 Collaboration Protocol Agreement 60

12.3 Countermeasure Technologies 60

12.3.1 Persistent Digital Signature 60

12.3.1.1 Signature Generation 61

12.3.2 Persistent Signed Receipt 62

12.3.3 Non-persistent Authentication..... 62

12.3.4 Non-persistent Integrity 62

12.3.5 Persistent Confidentiality..... 63

12.3.6 Non-persistent Confidentiality..... 63

12.3.7 Persistent Authorization..... 63

12.3.8 Non-persistent Authorization 63

12.3.9 Trusted Timestamp..... 63

13 Synchronous and Asynchronous Responses..... 66

14 References..... 67

14.1 Normative References..... 67

14.2 Non-Normative References 67

15 Disclaimer 69

16 Contact Information..... 70

Appendix A ebXMLHeader Schema and Data Type Definitions 72

A.1 Schema Definition 72

A.2 Data Type Definition 76

Appendix B Examples..... 77

Appendix C Communication Protocol Interfaces..... 78

C.1 HTTP 78

C.1.1 Asynchronous HTTP 78

C.1.2 Synchronous HTTP 79

C.1.3 Use of Error Codes..... 79

C.2 SMTP..... 80

A.1 81

C.3 Communication Errors during Reliable Messaging 81

Appendix D Request for MIME media type Application/Vendor Tree - vnd..... 82

Copyright Statement 84

1 4 Introduction

2 This is a draft standard for trial implementation. ~~The~~ This 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 a separate document. ~~It will be~~
7 ~~included in a later version of this specification or as an additional specification.~~ It SHALL either be
8 incorporated into a future version of this specification or referenced as an external specification
9 as deemed most suitable by the ebXML Transport, Routing and Packaging project team.

10 4.1 Summary of Contents of Document

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

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

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

20 This specification is organized around the following topics:

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

38 Appendices to this specification cover the following:

- 39 • Appendix A Schemas and DTD Definitions – This contains [XML Schema] and [XML]
40 Data Type Definitions for the ebXML Header document. Section A.1 is normative while
41 Section A.2 is non-normative.
- 42 • Appendix B Examples – This contains a non-normative sample message content
- 43 • Appendix C Communication Protocol Envelope Mappings – This normative appendix
44 describes how to transport ebXML Message Service compliant messages over [HTTP]
45 and [SMTP]

- 46 • Appendix D ~~Reliable Messaging Protocol Logic~~—this non-normative appendix provides
 47 ~~processing logic that describes the behavior of a Message Service Handler when sending~~
 48 ~~or receiving messages with reliable delivery~~Registration of MIME media type
 49 ~~Application/Vendor Tree—vnd~~ – This non-normative appendix contains the registration
 50 ~~information that was forwarded to IANA to register the MIME subtype vnd.eb+xml.~~

51 4.2 Document Conventions

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

55 The keywords MUST, MUST NOT, REQUIRED, SHALL, SHALL NOT, SHOULD, SHOULD NOT,
 56 RECOMMENDED, MAY, and OPTIONAL, when they appear in this document, are to be
 57 interpreted as described in RFC 2119 [Bra97] as quoted here:-

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

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

81 4.3 Audience

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

84 4.4 Caveats and Assumptions

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

87 4.5 Related Documents

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

- 89 • ebXML Message Services Requirements Specification [EBebXMLMSREQ] – defines
 90 the requirements of ~~the~~ these Message Services

- 91 • [ebXML Technical Architecture \[EBXMLTA\]](#) – defines the overall technical architecture
- 92 [for ebXML](#)
- 93 • [ebXML Technical Architecture Security Specification \[EBXMLTASEC\]](#) – defines the
- 94 security mechanisms necessary to negate anticipated, selected threats
- 95 • [ebXML Collaboration Protocol Profile and Agreement Specification \[EBebXMLTP\]](#)
- 96 (under development) - defines how one party can discover and/or agree upon the
- 97 information that party needs to know about another party prior to sending them a
- 98 message that complies with this specification
- 99 • [ebXML Message Service Interface Specification](#) (to be developed) - defines an
- 100 interface that may be used by software to interact with an ebXML Message Service
- 101 • [ebXML Registry Services Specification \[EBXMLRSS\]](#) – defines a registry service for
- 102 [the ebXML environment](#)

103 5 Design Objectives

104 ~~The design objectives of this specification are to define a Message Service (MS) to support XML~~
105 ~~based electronic business between small, medium and large enterprises. The design objectives of~~
106 ~~this specification are to define a wire format and protocol for a Message Service (MS) to support~~
107 ~~XML-based electronic business between small, medium, and large enterprises. While the~~
108 ~~specification has been primarily designed to support XML-based electronic business, the authors~~
109 ~~of the specification have made every effort to ensure that non-XML business information is fully~~
110 ~~supported.~~ This specification is intended to enable a low cost solution, while preserving a
111 vendor's ability to add unique value through added robustness and superior performance. It is the
112 intention of the Transport, Routing and Packaging Project Team to keep this specification as
113 straightforward and succinct as possible.

114 Every item in this specification will be prototyped by the ebXML Proof of Concept Team in order
115 to ensure the ~~clarity and accuracy of this specification~~clarity, accuracy and efficiency of this
116 specification.

117 6 System Overview

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

123 6.1 What the Message Service does

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

127 The ebXML Message Service is provided for environments requiring a robust, yet low cost
128 solution to enable electronic business. It is one of the three "infrastructure" components of ebXML
129 ~~that includes; the other two are~~ Registry/Repository [ebXMLRegRep], Collaboration Protocol
130 Profile/Agreement [ebXMLTP] and the ebXML Message Service.

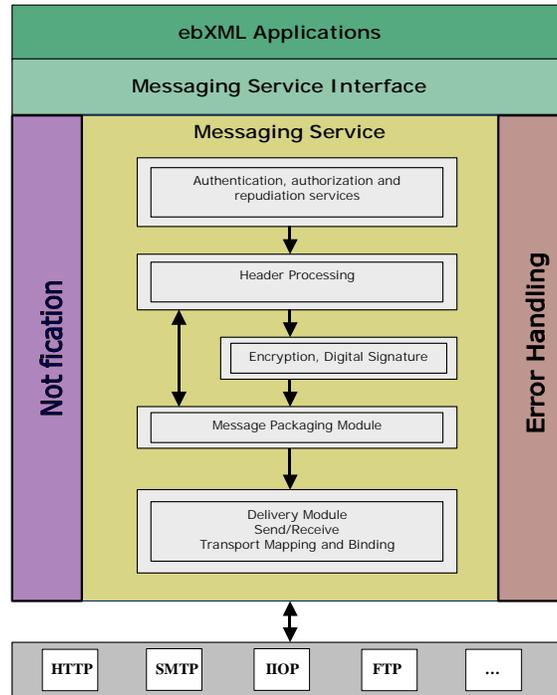
131 6.2 Message Service Overview

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

135 The following diagram depicts a logical arrangement of the functional modules that exist within
136 ~~the one possible implementation of the~~ ebXML *Messaging Services* architecture. These modules
137 are arranged in a manner to indicate their inter-relationships and dependencies.

- 138 • Header Processing - the creation of the ebXMLHeader document for the ebXML
139 Message uses input from the application, passed through the Message Service Interface,
140 information from the CPA that governs the message, and generated information such as
141 digital signature, timestamps and unique identifiers.
- 142 • Header Parsing - extracting or transforming information from a received ebXMLHeader
143 into a form that is suitable for processing by the MSH implementation.
- 144 • Security Services - digital signature creation and verification, authentication and
145 authorization. These services MAY be used by other components of the MSH including
146 the Header Processing and Header Parsing components.
- 147 • Reliable Messaging Services - handles the delivery and acknowledgment of ebXML
148 Messages sent with deliverySemantics of OnceAndOnlyOnce. The service includes
149 handling for persistence, retry, error notification and acknowledgment of messages
150 requiring reliable delivery.
- 151 • Message Packaging - the final enveloping of an ebXML Message (ebXMLHeader and
152 payload) into its MIME multipart/related container
- 153 • Error Handling - this component handles the reporting of errors encountered during MSH
154 or Application processing of a message as well as processing of received messages that
155 have an ErrorList element detailing an error reported by a foreign MSH on a message
156 previously sent by the MSH.
- 157 • Notification - <rb>add additional text here for description </rb>
- 158 • Message Service Interface - an abstract service interface that applications use to interact
159 with the MSH to send and receive messages and which the MSH uses to interface with
160 applications that handle received messages.

161



162

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

164 <DB>Diagram needs to be simplified and an explanation of these components needs to be
 165 provided. (Ralph & Chris)</DB>

166 7 Packaging Specification

167 7.1 Introduction

168 An ebXML Message consists of two parts:

- 169 • an outer Communication Protocol Envelope, such as HTTP or SMTP,
- 170 • an inner communication “protocol independent” ebXML Message Envelope, specified
- 171 using MIME multipart/related, that contains the two main parts of the Message:
 - 172 - an ebXML Header Container that is used to envelope one ebXML Header Document,
 - 173 - ~~an optional, single~~ at most one ebXML Payload Container that MUST be used to
 - 174 envelope the actual payload (transferred data) of the Message Communication
 - 175 Protocol Envelope (SMTP, HTTP, etc)

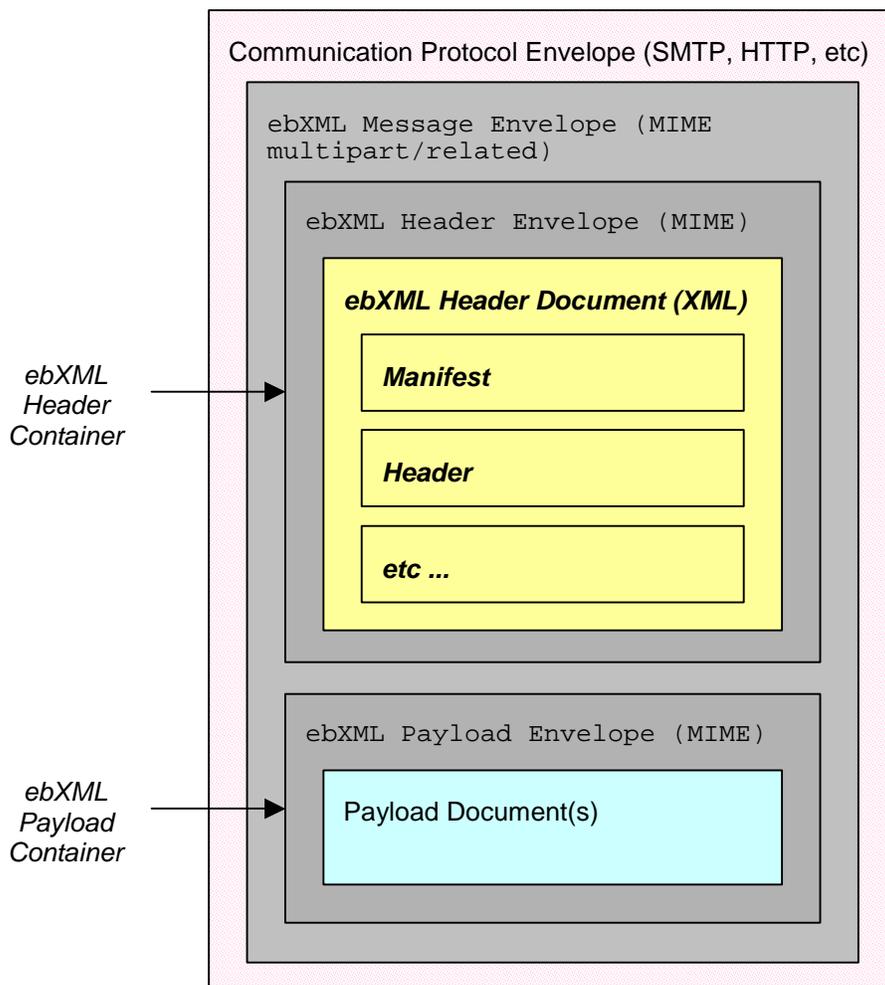


Figure 7-1 ebXML

176
177

Message Structure

178 7.1.1 ebXML Header Envelope and ebXML Payload Envelope

179 An ebXML Header Envelope and an ebXML Payload Envelope are constructed of standard,
180 MIME components.

181 The ebXML Header Envelope contains a single ebXMLHeader document (see section 8). The
 182 ~~ebXML Payload Envelope can contain any electronic data that can be transported within~~
 183 ~~MIME contents of the ebXML Payload are determined by the user of the ebXML service.~~

184 Any special considerations for the usage of the ebXML Message Envelope in HTTP and SMTP
 185 transports are described in Appendix C.

186 **7.1.2 MIME usage Conventions**

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

189 **7.2 ebXML Message Envelope**

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

193 **7.2.1 Content-Type**

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

```
196 Content-Type: multipart/related+
```

198 The MIME Content-Type header contains three attributes:

- 199 • type
- 200 • boundary
- 201 • version

202 **7.2.1.1 type Attribute**

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

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

210 **7.2.1.2 boundary Attribute**

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

```
215 boundary:="-----8760boundaryValueHere"
```

217 **7.2.1.3 version Attribute**

218 The MIME *version* attribute indicates the version of the ebXML Message Service Specification
 219 to which the *ebXML Message Envelope* conforms. All message headers SHOULD USE
 220 "0.930-92". For example:

```
221 version="0.930-92"
```

222 **7.2.2 ebXML Message Envelope Example**

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

224

```
225 Content-Type: multipart/related; type="application/vnd.eb+xml"; boundary:="-----  
226 8760boundaryValue";
```

227 7.3 ebXML Header Container

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

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

231 The following rules apply:

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

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

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

- 237 • Content-ID
- 238 • Content-Type

239 7.3.1 Content-ID

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

```
243 Content-ID: <2000-0722-161201-123456789@ebxmlhost.realmexample.com>  
244
```

245 7.3.2 Content-Type

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

- 248 • version
- 249 • charset

250 7.3.2.1 version Attribute

251 The MIME `version` attribute indicates the version of the ebXML Message Service Specification
252 to which the *ebXML Header Envelope* and *ebXML Header Document* conform. All message
253 headers MUST USE "0.93~~0.92~~". Future versions of this specification may require other values of
254 this attribute. However, the value specified here MUST match that specified in the `version`
255 attribute of the *ebXML Header Document* for all versions of this specification. For example:

```
256 version="0.930.92";  
257
```

258 7.3.2.2 charset Attribute

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

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

```
269 charset="UTF-8"
```

271 **7.3.3 ebXML Header Envelope Example**

272 The following represents an example of an *ebXML Header Envelope* and *ebXML Header*
 273 *Document*:

```
274
275 Content-ID: ebxmlheader-123@ebxmlhost.reexample.comalm --| MIME ebXML |
276 Content-Type: application/vnd.eb+xml; | Header Envelope |
277 version="0.930-92"; charset="UTF-8" - --| | ebXML |
278 | Header |
279 | Container |
280 <ebxMLHeader> -----| |
281 <Manifest>..... | XML ebxML Header |
282 </Manifest> | Document |
283 <Header>..... | |
284 </Header> | |
285 <Routing Header>..... | |
286 </Routing Header> | |
287 </ebxMLHeader> -----| |
```

287 A complete example of an *ebXML Header Container* is presented in Appendix B. That example
 288 includes the `charset` attribute and portions of an *XML Prolog* (see sect 8.1), ~~none-neither~~ of
 289 which is required to appear in an *ebXML Header Container* or *ebXML Header Document*.
 290 Appendix B also includes the outer *ebXML Message Envelope* and a complete (valid)
 291 **ebxMLHeader** element rather than the outline shown above.

292 **7.4 ebXML Payload Container**

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

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

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

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

301 ~~If an ebXML Payload Container is present, it MUST conform to MIME [RFC2045] and MUST~~
 302 ~~consist of:~~

- 303 ~~—a MIME header portion—the ebXML Payload Envelope, and~~
- 304 ~~• a content portion—the payload itself that may be of any valid MIME type.~~

305 If an ebXML Payload Container is present, it MUST conform to MIME [RFC2045] and MUST
 306 consist of a single payload MIME object that may be any valid MIME type including any of the
 307 MIME multipart/* types.

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

- 309 • Content-ID
- 310 • Content-Type

311 The *ebXML Message Service Specification* makes no provision, nor limits in any way the
 312 structure or content of payloads. ~~Payloads MAY be a simple plain-text object or complex nested~~

313 ~~multipart objects. This is the implementer's decision.~~ Payloads MAY be a simple-plain-text object
 314 or complex nested multipart objects. The specification of the structure and composition of
 315 payload objects is the prerogative of the organization that defines the business process or
 316 information exchange that uses the ebXML Message Service.

317 **7.4.1 Content-ID**

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

```
321 Content-ID: <2000-0722-161201-123456789@ebxmlhost.realmexample.com>
```

323 **7.4.2 Content-Type**

324 ~~The MIME Content-Type for an ebXML payload is determined by the implementer and is used~~
 325 ~~to identify the type of data contained in the content portion of the ebXML Payload Container. The~~
 326 ~~MIME Content-Type MUST conform to [RFC2045]. For example~~The MIME Content-Type for an
 327 ebXML Payload Container is used to specify the media type and subtype of data in the body of
 328 the ebXML Payload Container. The value of this MIME parameter is determined by the
 329 organization that defines the business process or information exchange. The value selected
 330 SHOULD be chosen from the list of registered MIME media types found at: [ftp://isi.edu/in-](ftp://isi.edu/in-notes/iana/assignments/media-types/)
 331 [notes/iana/assignments/media-types/](ftp://isi.edu/in-notes/iana/assignments/media-types/). The MIME Content-Type MUST conform to [RFC2045].
 332 For example:

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

335 **7.4.3 Example of an ebXML MIME Payload Container**

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

```
337 Content-ID: ebxmlpayload-123@ebxmlhost.realmdomainname.example.com-----|
338 ebXML MIME |
339 Content-Type: application/xml -----| Payload Envelope | ebXML
340 | | | Payload
341 <Invoice> -----| | Container
342 <Invoicedata>..... | Payload
343 </Invoicedata> |
344 </Invoice> -----| |
```

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

346 **7.5 Additional MIME Parameters**

347 Any MIME part described by this specification MAY contain additional MIME parameters in
 348 conformance with the [MIME RFC2045] specification. Implementations MAY ignore any MIME
 349 parameter not defined in this specification. Implementations MUST ignore any MIME parameter
 350 that they do not recognize.

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

353

354 **7.6 Reporting MIME Errors**

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

359 8 ebXML Header Document

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

- 362 • different software components are likely to be used to generate that header-element,
- 363 • the element is not always present,
- 364 • the structure of the header element might vary independently of the other header-
 365 elements, or
- 366 • the data contained in the header-element MAY need to be digitally signed separately
 367 from the other header-elements.

368 8.1 XML Prolog

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

```
372 <?xml version="1.0" encoding="UTF-8"?>
373 <!DOCTYPE ebXMLHeader SYSTEM "level1-10122000.dtd">
374 <ebXMLHeader>...</ebXMLHeader>
```

376 8.1.1 XML Declaration

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

381 8.1.2 Encoding Declaration

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

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

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

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

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

397 For example:

```
398 Content-Type:application/vnd.eb+eml; version "0.93"; charset="UTF-8"
399 <?xml version="1.0" encoding="UTF-8"?>
```

400 8.1.3 Standalone Document Declaration

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

404 ~~<DB>What do you do if the XML Recommendation is not met?</DB>~~

405 8.1.4 Document Type Declaration

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

```
409
410 <!DOCTYPE ebXMLHeader SYSTEM "level1-10122000.dtd">
```

411 ~~<DB>1. Do we need to mandate use of a DOCTYPE and then maybe remove it in a later version
412 of the spec when everyone is using XML Schema~~

413 ~~2. Do we we need to be prescriptive about what goes in the Prolog depending on whether we
414 have a DTD or XSD.(Saikat)</DB>~~

415 8.2 ebXMLHeader Element

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

418 8.2.1 ebXMLHeader attributes

419 There are two attributes defined for the **ebXMLHeader** element, they are as follows:

- 420 • Namespace (xmlns)
- 421 • Version

422 Additional namespace declarations and namespace-qualified attributes from foreign namespaces
423 MAY be added to support extensions to the **ebXMLHeader** document.

424 8.2.1.1 Namespace attribute

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

427 8.2.1.2 version attribute

428 The required **version** attribute indicates the version of the ebXML Message Service Specification
429 to which the *ebXML Header Document* conforms. Its purpose is to provide for future versioning
430 capabilities. All *ebXML Header Documents* MUST USE "0.930.92". Future versions of this
431 specification SHALL require other values of this attribute. However, the value specified here
432 MUST match that specified in the MIME *version* attribute of the *ebXML Header Envelope* for all
433 versions of this specification.

434 8.2.2 ebXMLHeader elements

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

- 436 • **Manifest** – an element that points to any data present either in the *ebXML Payload*
437 *Container* or elsewhere, e.g. on the web
- 438 • **Header** – a REQUIRED element that contains routing information for the message
439 (To/From, etc.) as well as other context information about the message
- 440 • **RoutingHeaderList** – an element that contains entries that identify the Message
441 Service Handler (MSH) that sent and should receive the message. This element can be
442 omitted.
- 443 • **ApplicationHeaders** – an element that can be used by a process or service to include
444 additional information that needs to be associated with the data in the *ebXML Payload*
445 ~~but is not contained within it that the MSH MUST make available to the application~~
446 processing the ebXML Payload Container
- 447 • **StatusData** – an element that is used by a MSH when responding to a request on the
448 status of a message that was previously received

- 449 • **ErrorList** – an element that contains a list of the errors that ~~have been found in a~~
450 ~~message are being reported against a previous message~~
- 451 • **Acknowledgment** – an element that is used by a ~~MSH to indicate that a message has~~
452 ~~been received~~ receiving MSH to acknowledge to the sending MSH that a previous
453 message has been received
- 454 • **Signature** – an element that contains a digital signature that conforms to [XMLDSIG] that
455 signs data associated with the message
- 456 • **#wildcard** - any namespace-qualified element content belonging to a foreign namespace

457 8.2.3 Combining Principal Header Elements

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

459 8.2.3.1 Manifest element

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

463 8.2.3.2 Header element

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

465 8.2.3.3 RoutingHeaderList element

466 The **RoutingHeaderList** element MAY be present in any message. It MUST be present if the
467 message is being sent reliably (see section 10) or over multiple hops (see section 8.5.3).

468 8.2.3.4 ApplicationHeaders element

469 The **ApplicationHeaders** element MAY be present on any message except a message that
470 contains an **ErrorList** element with a **highestSeverity** attribute set to **Error**.

471 8.2.3.5 StatusData element

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

- 473 • a **Manifest** element
- 474 • an **ErrorList** element with a **highestSeverity** attribute set to **Error**

475 8.2.3.6 ErrorList element

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

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

- 480 • a **Manifest** element
- 481 • an **ApplicationHeaders** element
- 482 • a **StatusData** element

483 8.2.3.7 Acknowledgment element

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

485 8.2.3.8 Signature element

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

487 8.2.3.9 #wildcard element content

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

492 Any namespace-qualified element added SHOULD include the global **mustUnderstand** attribute.
493 If the **mustUnderstand** attribute is NOT present, the default value implied is 'false'. If an
494 implementation of the MSH does not recognize the namespace of the element and the value of
495 the **mustUnderstand** attribute is 'true' then the MSH SHALL respond with a message that
496 includes an **errorCode** of **NotSupported** in an **Error** element as defined in section 8.8. If the
497 value of the **mustUnderstand** attribute is 'false' or if the **mustUnderstand** attribute is not present
498 then an implementation of the MSH MAY ignore the namespace-qualified element and its
499 content.

500 8.2.4 ebXMLHeader sample

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

```
502 <?xml version="1.0" encoding="UTF-8"?>
503 <ebXMLHeader xmlns="http://www.ebxml.org/namespaces/messageHeader" Version="0.930-92" >
504 <Manifest>...</Manifest>
505 <Header>...</Header>
506 <RoutingHeaderList>...</RoutingHeaderList>
507 </ebXMLHeader>
```

509 8.3 Manifest element

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

- 515 • to make it easier to directly extract a particular document associated with this *Message*,
- 516 • to enable a MSH to check the integrity of a *Message*
- 517 • to allow an application to determine whether it can process the payload without having to
518 parse it.

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

520 8.3.1 Reference element

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

- 522 • **Schema** - information about the schema(s) that define the instance document identified
523 in the parent **Reference** element
- 524 • **Description** - a textual description of the payload object referenced by the parent
525 **Reference** element
- 526 • **#wildcard** - any namespace-qualified element content belonging to a foreign namespace

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

532 The **Reference** element has the following attribute content in addition to the element content
533 described above:

- 534 • **id** - a REQUIRED XML ID for the **Reference** element,

- 535 • **xlink:type** - this attribute defines the element as being an XLINK simple link. It has a
536 fixed value of 'simple',
- 537 • **xlink:href** - this REQUIRED attribute has a value that is the URI of the payload object
538 referenced. It SHALL conform to the [XLINK] specification criteria for a simple link,
- 539 • **xlink:role** - this attribute identifies some resource that describes the payload object or its
540 purpose. If present, then it SHALL have a value that is a valid URI in accordance with the
541 [XLINK] specification,
- 542 • Any other namespace-qualified attribute MAY be present. A receiving MSH MAY choose
543 to ignore any foreign namespace attributes other than those defined above.

544 8.3.1.1 Schema element

545 If the item being referenced has schema(s) of some kind that describe it (e.g. an XML Schema,
546 DTD or a database schema), then the **Schema** element SHOULD be present as a child of the
547 **Reference** element. It provides a means of identifying the schema, and its version, that defines
548 the payload object identified by the parent **Reference** element. The Schema element contains the
549 following attributes:

- 550 • **location** - the REQUIRED URI of the schema
- 551 • **version** – a version identifier of the schema

552 8.3.1.2 Description element

553 The **Reference** element MAY contain zero or more **Description** elements. The **Description** is a
554 textual description of the payload object referenced by the parent **Reference** element. The
555 language of the description is defined by a REQUIRED **xml:lang** attribute. The **xml:lang** attribute
556 MUST comply with the rules for identifying languages specified in [XML]. This element is provided
557 to allow a human readable description of the payload object identified by the parent Reference
558 element. If multiple Description elements are present, each SHOULD have a unique **xml:lang**
559 attribute value. An example of a **Description** element follows.

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

561 8.3.1.3 #wildcard element

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

563 8.3.2 What References are Included in a Manifest

564 The designer of the ~~protocol or application~~ **business process or information exchange** that is using
565 ebXML Messaging decides what payload data is referenced by the Manifest and the values to be
566 used for **xlink:role**.

567 8.3.3 Manifest Validation

568 If an **xlink:href** attribute contains a URI that is a content id (URI scheme "cid") then a MIME
569 part with that `content-id` MUST be present in the *ebXML Payload Container* of the message.
570 If it is not, then the error SHALL be reported to the *From Party* with an **errorCode** of
571 **MimeProblem** and a **severity** of **Error**.

572 If an **xlink:href** attribute contains a URI that is not a content id (URI scheme "cid") and that URI
573 cannot be resolved, then it is an implementation decision on whether to report the error. If the
574 error is to be reported, then it SHALL be reported to the *From Party* with an **errorCode** of
575 **MimeProblem** and a **severity** of **Error**.

576 8.3.4 Manifest sample

577 The following fragment demonstrates a typical **Manifest** for a message with a single payload
578 MIME body part:
579

```

580 <Manifest id="Manifest">
581   <Reference id="pay01"
582     xlink:href="cid:payload-1"
583     xlink:role="http://regrep.org/gci/purchaseOrder">
584     <Description>Purchase Order for 100,000 widgets</Description>
585     <Schema location="http://regrep.org/gci/purchaseOrder/po.xsd"
586       version="1.0"/>
587   </Reference>
588 </Manifest>

```

589 8.4 Header element

590 The **Header** element immediately follows the **Manifest** element. It is REQUIRED in all
 591 **ebXMLHeader** documents. The **Header** element is a composite element comprised of the
 592 following subordinate elements:

- 593 • **From**
- 594 • **To**
- 595 • **CPAId**
- 596 • **ConversationId**
- 597 • **Service**
- 598 • **Action**
- 599 • **MessageData**
- 600 • **QualityOfServiceInfo**
- 601 • **SequenceNumber**
- 602 • **Description**
- 603 • **#wildcard**

604 ~~The Header attribute MAY have an attribute: **id** that is of type XML ID.~~

605 8.4.1 From and To elements

606 The REQUIRED **From** element identifies the *Party* that originated the message. The REQUIRED
 607 **To** element identifies *Party* that is the intended recipient of the message. Both **To** and **From** can
 608 be logical identifiers such as a DUNS number or identifiers that also imply a physical location,
 609 such as an email address.

610 The **From** and the **To** elements have a single child element, **PartyId**.

611 The **PartyId** element has a single attribute, **type** and content that is a string value. If the **type**
 612 attribute is present, then it MUST be a URN. It indicates the domain of names to which the string,
 613 in the content of the **From** or **To** element, belongs.

614 If the **PartyId type** attribute is not present, the content of the **PartyId** element MUST be an URI
 615 [RFC 2396] otherwise report an error (see section 11) with **errorCode** set to **Inconsistent** and
 616 **severity** set to **error**. It is strongly RECOMMENDED that the content be an URN.

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

```

619
620 <From>
621   <PartyId type="urn:duns.com">1234567890123</PartyId>
622 </From>
623 <To>
624   <PartyId>smtp:joe@example.com</PartyId>
625 </To>

```

626 8.4.2 CPAId element

627 The REQUIRED **CPAId** element is a string that identifies the *Collaboration Protocol Agreement*
 628 (CPA) that governs the processing of the message. ~~It MUST be an identifier that~~ The identifier

629 ~~MUST be~~ unique within the ~~combination of the From and To Parties~~ domain of the names chosen
630 ~~by the Parties.~~

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

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

636 8.4.3 ConversationId element

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

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

644 The value used for a **ConversationId** is implementation dependent.

645 Note that implementations are free to choose how they will identify and store conversational state
646 related to a specific Conversation. Implementations SHOULD provide a facility for mapping
647 between their identification schema and a ConversationId generated by another implementation.

648 ~~<DB>I think that this last sentence should be deleted as it requiring implementation behavior that~~
649 ~~is an implementation decision.</DB>~~

650 8.4.4 Service element

651 The REQUIRED **Service** element identifies the service that acts on the message. It is specified
652 by the designer of the service. The designer of the service may be:

- 653 • a standards organization, or
- 654 • an individual or enterprise

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

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

658 8.4.4.1 type attribute

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

662 If the **type** attribute is not present, the content of the **Service** element MUST be a URI [RFC
663 2396]. If it is not a URI then report an error with an **errorCode** of **Inconsistent** and a **Severity** of
664 **Error** (see section 11).

665 8.4.4.2 ebXML Message Service namespace

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

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

668 8.4.5 Action element

669 The REQUIRED **Action** element identifies a process within a **Service** that processes the
670 Message. **Action** SHALL be unique within the **Service** in which it is defined.

671 8.4.6 MessageData element

672 The REQUIRED **MessageData** element provides a means of uniquely identifying an ebXML
673 Message. It contains the following ~~three~~four elements:

- 674 • **MessageId**
- 675 • **Timestamp**
- 676 • **RefToMessageId**
- 677 • **TimeToLive**

678 8.4.6.1 MessageId element

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

682 8.4.6.2 Timestamp element

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

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

687 8.4.6.3 RefToMessageId element

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

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

693 For Acknowledgment Messages, the **RefToMessageId** element is REQUIRED, and its value
694 MUST be the **MessageId value** of the ebXML Message being acknowledged. See also sections
695 8.2.3.7 and 10.

696 8.4.6.4 TimeToLive element

697 The TimeToLive element indicates the time by which a message should be delivered to and
698 processed by the To Party.

699 In this context, the TimeToLive has expired if the time of the internal clock of the MSH that
700 receives a message is greater than the value of TimeToLive for the message.

701 When setting a value for TimeToLive it is RECOMMENDED that the From Party takes into
702 account the accuracy of its own internal clocks as well as the MSH TimeAccuracy parameter for
703 the Receiving MSH (see section 10.6.5.3) that indicates the accuracy to which a MSH will keep
704 its internal clocks. How a MSH ensures that its internal clocks are kept sufficiently accurate is an
705 implementation decision.

706 If the TO Party's MSH receives a message where TimeToLive has expired, it SHALL send a
707 message to the From party MSH, reporting that the TimeToLive of the message has expired.
708 This message SHALL be comprised of:

- 709 • A Payload that consists of the ebXML message that expired;
- 710 • An ErrorList containing an Error that has the errorCode attribute set to
711 TimeToLiveExpired, and the severity attribute set to Error.

712 8.4.7 QualityOfServiceInfo element

713 The **QualityOfServiceInfo** element identifies the quality of service with which the message is
714 delivered. This element has ~~four~~four attributes:

- 715 • **deliverySemantics**
- 716 • **messageOrderSemantics**
- 717 • **deliveryReceiptRequested**
- 718 • **syncReplyMode**, and
- 719 ~~—timeToLive.~~

720 The **QualityOfServiceInfo** element ~~is~~MAY be present if any of the attributes within the element
721 need to be set to their non-default value.

722 8.4.7.1 deliverySemantics attribute

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

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

- 727 • **OnceAndOnlyOnce**. The message must be sent using a **reliableMessagingMethod**
728 that will result in the application or other process at the *To Party* receiving the message
729 once and only once
- 730 • **BestEffort** The reliable delivery semantics are not ~~specified~~used. In this case the value
731 of **reliableMessagingMethod** is ignored.

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

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

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

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

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

745

746 ~~<QualityOfServiceInfo deliverySemantics="OnceAndOnlyOnce"/>~~

747 8.4.7.1 messageOrderSemantics attribute

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

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

- 753 • **Guaranteed**. The messages are passed to the receiving application in the order which
754 the sending application specified.
- 755 • **NotGuaranteed** The messages may be passed to the receiving application in different
756 order from the order which sending application specified.

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

759 If **messageOrderSemantics** is set to **Guaranteed** then the *To Party* MSH MUST correct invalid
760 order of messages using the value of **SequenceNumber** in the conversation specified the
761 **ConversationId**. The **Guaranteed** semantics can be set only when **deliverySemantics** is
762 **OnceAndOnlyOnce**. If **deliverySemantics** is not **OnceAndOnlyOnce** then report the error to
763 the *From Party* with an **errorCode** of **Inconsistent** and a **severity** of **Error** (see section **Error!**
764 **Reference-source-not-found**, 10).

765 If **deliverySemantics** is set to **NotGuaranteed**, then the *To Party* MSH does not need to correct
766 invalid order of messages. If the *To Party* is unable to support the type of
767 **MessageOrderSemantics** requested, then the *To Party* MUST report the error to the *From Party*
768 using an **ErrorCode** of **NotSupported** and a **Severity** of **Error**. A sample of
769 **messageOrderSemantics** follows.

770
771

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

772 8.4.7.2 DeliveryReceiptRequested attribute

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

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

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

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

784 Valid values for **deliveryReceiptRequested** are:

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

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

790 If the *To Party* MSH can produce the Delivery Receipt of the type requested, then it MUST return
791 to the *From Party* on the message just received, a message containing an **Acknowledgment**
792 element with the value of the **type** attribute set to **DeliveryReceipt**.

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

795 8.4.7.3 syncReplyMode attribute

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

- 798 • **True** which indicates that the MSH that receives the message MUST get the message
799 processed by the application or other process that needs to process it before the MSH
800 sends any response to the original message, or
- 801 • **False** which indicates that an acknowledgment to the message MAY be sent separately
802 before processing of the message by the application or other process.

803 The default value is **False**.

804 **8.4.7.4 TimeToLive attribute**

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

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

811 ~~How a MSH ensures that its internal clocks are kept sufficiently accurate is an implementation~~
812 ~~decision.~~

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

- 814 ~~– send a Message to the From Party MSH, reporting that the **TimeToLive** of the message~~
- 815 ~~has passed~~
- 816 ~~– NOT forward the message to another MSH or application/other system that should receive~~
- 817 ~~the message.~~

818 ~~The message reporting the error MUST contain an **ErrorCode** set to **TimeToLiveExpired**, and a~~
819 ~~**severity** attribute set to **Error**~~

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

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

824 **8.4.8 SequenceNumber element**

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

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

836 If the implementation defined limit for saved out-of-sequence messages is reached, then the To
837 Party MSH MUST indicate a delivery failure to the From Party MSH with **errorCode** set to
838 **DeliveryFailure** and **severity** set to **Error** (see section ~~Error! Reference source not found,~~ 11).

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

- 844 1) First message from the within the Conversation
- 845 2) First message after resetting Sequence Number information by the From Party MSH
- 846 3) First message after wraparound (next value after 99999999)

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

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

851 When the Sequence Number is set to “0” because of 1 or 2 above, the **Status** attribute of the
 852 messages MUST be set to “Reset”. In all other cases, including 3 above, the **Status** attribute
 853 MUST be set to “Continue”. Before the From Party resets the SequenceNumber of a
 854 *Conversation*, the Sender MUST wait for receiving of all the *Acknowledgement Messages* for
 855 Messages previously sent for the *Conversation*. Only when all the sent Messages are
 856 acknowledged, can the From Party reset the **SequenceNumber**. An example of a Sequence
 857 Number follows.

```
858  
859 <SequenceNumber Status="Reset">0</SequenceNumber>
```

860 8.4.9 Description element

861 The **Description** element is present zero or more times as a child element of the Header. Its
 862 purpose is to provide a human readable description of the purpose or intent of the message. The
 863 language of the description is defined by a required **xml:lang** attribute. The **xml:lang** attribute
 864 MUST comply with the rules for identifying languages specified in [XML]. Each occurrence
 865 SHOULD have a different value for **xml:lang**.

866 8.4.10 #wildcard element

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

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

872 8.4.11 Header sample

873 The following fragment demonstrates the structure of the **Header** element of the **ebXMLHeader**
 874 document:

```
875  
876 <Header id="N01">  
877   <From>...</example.com</From>  
878   <To type="userType">...</To>  
879   <CPAId>http://www.ebxml.org/cpa/123456</CPAId>  
880   <ConversationId>987654321</ConversationId>  
881   <Service type="myservicetypes">QuoteToCollect</Service>  
882   <Action>NewPurchaseOrder</Action>  
883   <MessageData>  
884     <MessageId>UUID-2</MessageId>  
885     <Timestamp>20000725T121905.000Z</Timestamp>  
886     <RefToMessageId>UUID-1</RefToMessageId>  
887   </MessageData>  
888  
889   <QualityOfServiceInfo  
890     deliverySemantics="OnceAndOnlyOnce"  
891     deliveryReceiptRequested="Signed"/>  
892   </QualityOfServiceInfo deliverySemantics="BestEffort"/>  
893 </Header>
```

894 8.5 RoutingHeaderList element

895 A **RoutingHeaderList** element consists of one or more **RoutingHeader** elements. Exactly one
 896 **RoutingHeader** is appended to the **RoutingHeaderList**, following any pre-existing
 897 **RoutingHeader** before transmission of a message over a data communication protocol.

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

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

901 **8.5.1 Routing Header Element**

902 The **RoutingHeader** element contains information about a single transmission of a message
 903 between two Parties. If a message traverses multiple hops by passing through ~~some type of~~
 904 ~~intermediate system between the From Party and the To Party, then each transmission over~~
 905 ~~each one or more intermediate MSH nodes as it travels between the From party MSH and the To~~
 906 ~~Party MSH, then each transmission over each successive "hop"~~ results in the addition of a new
 907 Routing Header element.

908 The **RoutingHeader** element is a composite element comprised of the following subordinate
 909 elements:

- 910 • **SenderURI**
- 911 • **ReceiverURI**
- 912 • **ErrorURI**
- 913 • **Timestamp**
- 914 • **SequenceNumber**
- 915 • **#wildcard**

916 The RoutingHeader element MAY contain either or both of the following attributes:

- 917 • **reliableMessagingMethod**
- 918 • **intermediateAckRequested**

919 **8.5.1.1 reliableMessagingMethod attribute**

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

- 922 • ebXML
- 923 • Transport

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

926 **8.5.1.2 intermediateAckRequested attribute**

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

- 929 • Signed
- 930 • Unsigned
- 931 • None

932 The default implied value for this attribute is "None". Refer to section 10.1.2 for discussion of the
 933 use of this attribute.

934

935 ~~**[These attributes are added to support the prototype Reliable Messaging content in this**~~
 936 ~~**specification—see note at beginning of section 10]**~~

937 **8.5.1.18.5.1.3 SenderURI element**

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

- 941 • responds to an earlier message
- 942 • acknowledges an earlier message

- 943 • reports an error in an earlier message.

944 **8.5.1.28.5.1.4 ReceiverURI element**

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

947 **8.5.1.38.5.1.5 ErrorURI element**

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

950 **8.5.1.48.5.1.6 Timestamp element**

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

953 **8.5.1.58.5.1.7 SequenceNumber element**

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

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

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

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

967 **8.5.1.68.5.1.8 #wildcard element**

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

969 **8.5.1.7reliableMessagingMethod attribute**

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

972 —ebXML

973 —Transport

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

976 **8.5.1.8intermediateAckRequested attribute**

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

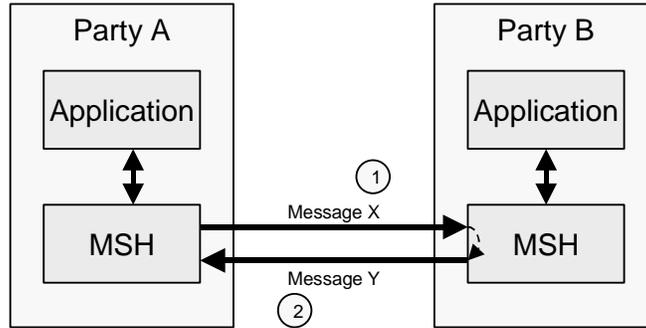
979 —Signed

980 —UnSigned

981 ~~—None~~
 982 ~~The default implied value for this attribute is “None”. Refer to section 10.1.2 for discussion of the~~
 983 ~~use of this attribute.~~

984 **8.5.2 Single Hop Routing Header Sample**

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



986
 987 **Figure 8-1 Single Hop Message**

988 The content of the corresponding messages could include:

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

```

990 <Header id="...">
991   <From>urn:myscheme.com:id:PartyA-id</From>
992   <To>urn:myscheme.com:id:PartyB-id</To>
993   <ConversationId>219cdj89dj2398djfjn</ConversationId>
994   ...
995   <MessageData>
996     <MessageId>29dmridj103kvna</MessageId>
997     ...
998   </MessageData>
999   ...
1000 </Header>
1001 <RoutingHeaderList id="...">
1002   <RoutingHeader>
1003     <SenderURI>url:PartyA.com/PartyAMsh</SenderURI>
1004     <ReceiverURI>url:PartyB.com/PartyBMSH</ReceiverURI>
1005     <Timestamp>20001216T21:19:35.145Z-8</Timestamp>
1006   </RoutingHeader>
1007 </RoutingHeaderList>
    
```

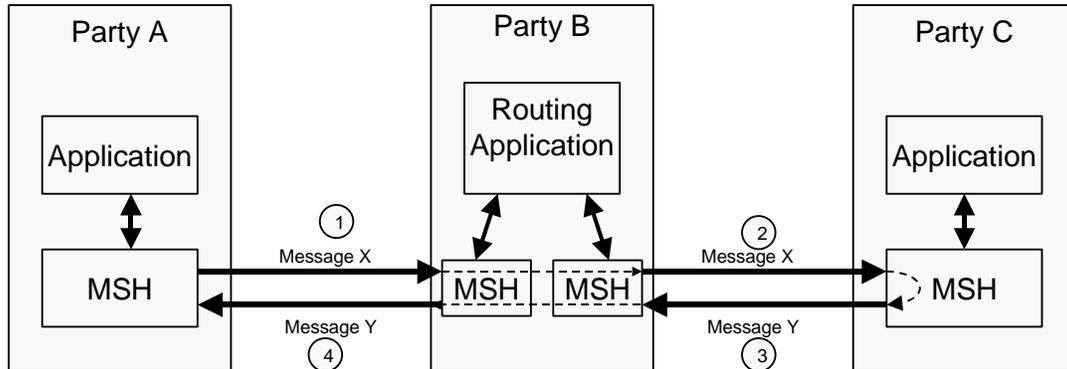
1009 ~~—Transmission 2 - Message Y From Party B To Party A~~

```

1010 <Header id="...">
1011   <From>urn:myscheme.com:id:PartyB-id</From>
1012   <To>urn:myscheme.com:id:PartyA-id</To>
1013   <ConversationId>219cdj89dj2398djfjn</ConversationId>
1014   ...
1015   <MessageData>
1016     <MessageId>eis99dk4mvzlgasi</MessageId>
1017     <RefToMessageId>29dmridj103kvna</RefToMessageId>
1018     ...
1019   </MessageData>
1020   ...
1021 </Header>
1022 <RoutingHeaderList id="...">
1023   <RoutingHeader>
1024     <SenderURI>url:PartyA.com/PartyAMsh</SenderURI>
1025     <ReceiverURI>url:PartyB.com/PartyBMSH</ReceiverURI>
1026     <Timestamp>20001216T21:20:05.274Z-6</Timestamp>
1027   </RoutingHeader>
1028 </RoutingHeaderList>
    
```

1029 **8.5.3 Multi-hop Routing Header Sample**

1030 Multi-hop messages are not sent directly from one party to another, instead they are sent via an
 1031 intermediate party. This is illustrated by the diagram below.



1032

1033 **Figure 8-2 Multi-hop Message**

1034 The content of the corresponding messages could include:

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

1035
1036
1037
1038
1039
1040
1041
1042
1043
1044
1045
1046
1047
1048
1049
1050
1051
1052
1053

```
<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

1054
1055
1056
1057
1058
1059
1060
1061
1062
1063
1064
1065
1066
1067
1068
1069
1070
1071
1072
1073
1074
1075
1076
1077

```
<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>
```

1078 ~~Message Y would be similar to Message X except that the direction of transmission is reversed.~~

1079 8.6 ApplicationHeaders Element

1080 The **ApplicationHeaders** element supports the extension of an ebXML Message through the
1081 inclusion of additional XML elements that belong to a foreign namespace, as child elements of
1082 the **ApplicationHeaders** element.

1083 Any additional element content MUST be namespace-qualified in accordance with
1084 [XMLNamespaces].

1085 An MSH implementation MUST make the information content of the **ApplicationHeaders**
1086 element available to the application or application services layer of software. How this is done is
1087 an implementation decision but conformance to the ebXML Service Interface specification (to be
1088 defined) is ~~recommended~~**RECOMMENDED**.

1089 ~~The **ApplicationHeaders** element has a single attribute called **mustUnderstand**. This attribute
1090 has two possible values **true** and **false**. The default value for the **mustUnderstand** attribute is
1091 **false**.~~

1092 ~~An **ApplicationHeaders** element that has a **mustUnderstand** set to a value of **true** means that a
1093 receiving MSH MUST be capable of understanding the meaning of the namespace-qualified
1094 element content. If the content is not understood, the receiving MSH MUST respond with a
1095 message that includes an **errorCode** of **NotSupported** in an **Error** element as defined in section
1096 8.8.~~

1097 8.6.1 ApplicationHeaders sample

```
1098 <ApplicationHeaders mustUnderstand="true">
1099   <foo:ProprietaryStuff
1100     xmlns:foo="http://www.example.com/ebxml-msh-extensions">...
1101   </foo:ProprietaryStuff>
1102 </ApplicationHeaders>
```

1103 8.7 StatusData Element

1104 The **StatusData** element is used by one MSH to respond to a request on the status of the
1105 processing of a message that was previously sent (see also section 9.1).

1106 The **StatusData** element consists of the following elements and attributes:

- 1107 • a **RefToMessageld** element that contains the **MessageId** of the message whose status
1108 is being reported
- 1109 • a **Timestamp** element. This contains the time that the message, whose status is being
1110 reported, was received. This MUST be omitted if the message whose status is being
1111 reported is **NotRecognized** or the request was **Unauthorized**
- 1112 • a **ForwardURI** element. This MUST only be present if **messageStatus** is set to
1113 **Forwarded**. If present it indicates the URI of the **ReceiverURI** to which the message was
1114 forwarded
- 1115 • a **messageStatus** attribute that is set to one of the following values:
 - 1116 - **Unauthorized** – the Message Status Request is not authorized or accepted
 - 1117 - **NotRecognized** – the message identified by the **RefToMessageld** element in the
1118 **StatusData** element is not recognized
 - 1119 - **Received** – the message identified by the **RefToMessageld** element in the
1120 **StatusData** element has been received by the MSH, but has not been processed by
1121 an application or forwarded to another MSH
 - 1122 - **Processed** – the message identified by the **RefToMessageld** element in the
1123 **StatusData** element has been received by the MSH for the To Party on the original
1124 message, and has been passed to the application or other process that is to handle it

1125 - **Forwarded** – the message identified by the **RefToMessageId** element in the
1126 **StatusData** element has been received by the MSH, and has been forwarded to
1127 another MSH

1128 8.8 ErrorList Element

1129 The existence of an **ErrorList** element indicates that the message that is identified by the
1130 **RefToMessageId** in the header has an error.

1131 The **ErrorList** element consists of one or more **Error** elements and the following two attributes:

- 1132 • **id** attribute
- 1133 • **highestSeverity** attribute

1134 If there are no errors to be reported then the **ErrorList** element MUST NOT be present.

1135 8.8.1 id attribute

1136 The **id** attribute uniquely identifies the **ErrorList** element within the document.

1137 8.8.2 highestSeverity attribute

1138 The **highestSeverity** attribute contains the highest severity of any of the **Error** elements.
1139 Specifically, if any of the **Error** elements has a **severity** of **Error** then **highestSeverity** must be
1140 set to **Error** otherwise set **highestSeverity** to **Warning**.

1141 8.8.3 Error element

1142 An **Error** element consists of the following attributes:

- 1143 • **codeContext**
- 1144 • **errorCode**
- 1145 • **severity**
- 1146 • **location**
- 1147 • **xml:lang**
- 1148 • **errorMessage**
- 1149 • **softwareDetails**

1150 8.8.3.1 codeContext attribute

1151 The REQUIRED **codeContext** attribute identifies the namespace or scheme for the **errorCodes**.
1152 It MUST be a URI. Its default value is **http://www.ebxml.org/messageServiceErrors**. If it is
1153 does not have the default value then it indicates that an implementation of this specification has
1154 used its own **errorCodes**.

1155 Use of non ebXML values for **errorCodes** is NOT RECOMMENDED. In addition, an
1156 implementation of this specification MUST NOT use its own **errorCodes** if an existing **errorCode**
1157 as defined in section 8.8.5 has the same or very similar meaning.

1158 8.8.3.2 errorCode attribute

1159 The required **errorCode** attribute indicates the nature of the error in the *message in error*. Valid
1160 values for the **errorCode** and a description of the code's meaning are given in section 8.8.5.

1161 8.8.3.3 severity attribute

1162 The required **severity** attribute indicates the severity of the error. Valid values are:

- 1163 • **Warning** - This indicates that although there is an error, other messages in the
1164 conversation will still be generated in the normal way.

- 1165 • **Error** - This indicates that there is an unrecoverable error in the message and no further
1166 messages will be generated as part of the conversation.

1167 **8.8.3.4 location attribute**

1168 The **location** attribute points to the part of the message that is in error.

1169 If an error exists in the ebXML Header document and the document is “well formed” (see [XML]),
1170 then the content of the **location** attribute MUST be an [XPointer].

1171 If the ebXML Header document is not “well formed” then the location attribute MUST be omitted.

1172 If the error is associated with the MIME envelope that wraps the ebXML Header Document and
1173 the ebXML Payload, then **location** id contains the content-id of the MIME part that is in error, in
1174 the format cid:23912480wsr, where the text after the “:” is the value of the MIME part’s content-
1175 id.

1176 The **location** attribute MUST NOT be used to point to errors inside the ebXML Payload Container
1177 as the method of reporting errors in the ebXML Payload Container is application dependent.

1178 **8.8.3.5 errorMessage attribute**

1179 The **errorMessage** attribute provides a narrative description of the error in the language defined
1180 by the **xml:lang** attribute. Typically, it will be the message generated by the XML parser or other
1181 software that is validating the message. This means that the value of the attribute is defined by
1182 the vendor/developer of the software, that generated the Error element.

1183 The **xml:lang** must comply with the rules for identifying languages specified in [XML].

1184 The **errorMessage** attribute MAY be omitted.

1185 <DB>Do we want to allow multiple errorMessage elements in different languages, e.g. so that if
1186 you send a message to Switzerland you could send it in French, German and Italian?</DB>

1187 **8.8.3.6 softwareDetails attribute**

1188 The **softwareDetails** attribute contains a value that is set by the vendor/developer of the software
1189 that generated the **Error** element. It SHOULD contain data that enables the vendor/developer as
1190 well as the recipient of the message to identify the precise location in their software and the set of
1191 circumstances that caused the software to generate a *message reporting the error*. It is
1192 RECOMMENDED that this element include plain text separated by punctuation to identify:

- 1193 • the name of the software vendor;
- 1194 • the name, version and release number of the software that generated the ebXML Error
1195 Document
- 1196 • the part of the software that caused the error to be generated that can be used by the
1197 Software Vendor to identify the circumstances that caused the error

1198 If any part of the **softwareDetails** attribute contains text that is readable by a human, then it
1199 SHOULD be in the language identified by **xml:lang**.

1200 **8.8.4 Examples**

1201 An example of an **ErrorList** element is given below.

```
1202 <ErrorList id='3490sdo9', highestSeverity="error">
1203   <Error errorCode='UnableToParse', severity="Error", location=cid:21398adhiwqe, xml:lang="us-
1204   en", errorMessage='XSD parser error - document not parsable', softwareDetails='Software
1205   Development Corp.; ebXML Connector!!! v2.7, build 2.7313; Ref HA' />
1206   <Error ... />
1207 </ErrorList>
```

1209 **8.8.5 errorCode values**

1210 This section describes the **ErrorCodes** (see section 8.8.3.2) that are used in a *message*
 1211 *reporting an error*. They are described in a table with three headings:

- 1212 • the first column contains the value to be used as an **errorCode**, e.g. **UnableToParse**
- 1213 • the second column contains a "Short Description" of the **errorCode**. Note that this
 1214 narrative MUST NOT be used in the **errorMessage** attribute.
- 1215 • the third columns contains a "Long Description" that provides an explanation of the
 1216 meaning of the error and provides guidance on when the particular **ErrorCode** should be
 1217 used.

1218 It is RECOMMENDED that implementers of software that conforms to this specification make
 1219 available to a user that is being informed of the error: the value of the **errorCode**, the "Short
 1220 Description" and optionally the "Long Description".

1221 It is also RECOMMENDED that the "Short Description" and the "Long Description" are translated
 1222 into the preferred language of the user if this is known.

1223 **8.8.6 Reporting Errors in the ebXML Header Document**

1224 The following list contains error codes that can be associated with the *ebXML Header Document*:
 1225

Error Code	Short Description	Long Description
UnableToParse	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".
ValueNotRecognized	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
NotSupported	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"> • is consistent with the rules and constraints contained in this specification, but • is not supported by the ebXML Message Service that is processing the message.
Inconsistent	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.
OtherXml	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.

1226 **8.8.7 Non-XML Document Errors**

1227 The following are error codes that identify errors that are not associated with the ebXML Header
 1228 Document:

Error Code	Short Description	Long Description
MessageTooLarge	Message too large	The message is too large to be processed by the ebXML Message Service.
MimeProblem	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"> • 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 • 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
DeliveryFailure	Message Delivery Failure	A message has been received that either probably or definitely could not be sent to its next destination. Note that if severity is set to Warning then there is a small probability that the message was delivered.
TimeToLiveExpired	Message Time To Live Expired	A message has been received that arrived after the time specified in the TimeToLive element of the Header element
SecurityFailure	Message Security Checks Failed	Validation of signatures or checks on the authenticity or authority of the sender of the message have failed.
Unknown	Unknown Error	Indicates that an error has occurred that is not covered explicitly by any of the other errors. The errorMessage attribute should be used to indicate the nature of the problem.

1229 **8.9 Acknowledgment Element**

1230 The Acknowledgment element is an optional element that is used by one Message Service
 1231 Handler to indicate that another Message Service Handler has received a message.

1232 For clarity two terms are defined:

- 1233 • *message being acknowledged*. This is the Message that is has been received by a MSH
 1234 that is now being acknowledged
- 1235 • *acknowledgment message*. This is the message that acknowledges that the *message*
 1236 *being acknowledged* has been received.

1237 The *message being acknowledged* is identified by the **RefToMessageId** contained in the
 1238 **MessageData** element contained within the **Header** Element of the acknowledgment message
 1239 containing the value of the **MessageId** of the message being acknowledged.

1240 The **Acknowledgment** element consists of the following:

- 1241 • a **Timestamp** element
- 1242 • a **From** element
- 1243 • a **type** attribute
- 1244 • a **signed** attribute

1245 8.9.1 Timestamp element

1246 The **Timestamp** element is a value representing the time that the *message being acknowledged*
1247 was received by the Party generating the *acknowledgment message*. It must conform to [ISO-
1248 8601]. <DB>Do we make this conform to XML Schema *timeInstant*</DB>

1249 8.9.2 From element

1250 This is the same element as the **From** element within **Header** element (see section 8.4.1).
1251 However, when used in the context of an Acknowledgment Element, it contains the identifier of
1252 the *Party* that is generating the *acknowledgment message*.

1253 If the **From** element is omitted then the *Party* that is sending the element is identified by the **From**
1254 element in the **Header** element.

1255 8.9.3 type attribute

1256 The **type** attribute indicates who sent the *acknowledgment message*. It MUST contain either:
1257

- 1258 • **DeliveryReceipt** - indicates that the *acknowledgment message* was generated by the *To*
1259 *Party* identified by the **To** element of the *message being acknowledged*, or
- 1260 • **IntermediateAck** - indicates that the *acknowledgment message* was generated by a
1261 *Party* that is not the *To Party* identified by the **To** element of the *message being*
1262 *acknowledged*. Typically this will be a *Party* that has received the message and is
1263 forwarding it to either the *To Party* or another *Party* with the intention that the message is
sent to the *To Party*.

1264 The default value for **type** is **DeliveryReceipt**.

1265 8.9.4 signed attribute

1266 The **signed** attribute indicates whether the *acknowledgment message* is digitally signed. It MUST
1267 contain either:

- 1268 • **True** - indicates that the *acknowledgment message* is digitally signed, or
- 1269 • **False** - indicates that the *acknowledgment message* is not digitally signed

1270 The default value for **signed** is **False**.

1271 See section 12 for details on what should be signed and how a signature that signs an
1272 *acknowledgment message* should be checked.

1273 8.10 Signature Element

1274 TBD An ebXML Message may be digitally signed to provide security countermeasures. Zero or
1275 more Signature elements, belonging to the [XMLDSIG] defined namespace MAY be present in an
1276 ebXMLHeader. The Signature element MUST be namespace qualified in accordance with
1277 [XMLDSIG]. The structure and content of the Signature element MUST conform to the
1278 [XMLDSIG] specification. If there are more than one Signature elements contained within the
1279 ebXMLHeader, the first MUST represent the digital signature of the ebXML Message as signed
1280 by the From Party MSH in conformance with section 12. Additional Signature elements MAY be
1281 present, but their purpose is undefined by this specification.

1282 Refer to section 12 for a detailed discussion on how to construct the Signature element when
1283 digitally signing an ebXML Message.

1284 9 Message Service Handler Services

1285 *[The Message Service Handler Services section has not been agreed to by the*
1286 *membership of the TRP Project Team; however, it is being included to provide a basis for*
1287 *POC developers of MSH implementations. Implementers MUST be prepared for some*
1288 *change to the content of this section.]*

1289 The Message Service Handler MUST support two services that are designed to help provide
1290 smooth operation of a Message Handling Service implementation:

- 1291 • Message Status Request
- 1292 • Message Service Handler Ping

1293 Each service is described below:

1294 9.1 Message Status Request Service

1295 The Message Status Request Service consists of the following:

- 1296 • sending a Message Status Request message to a Message Service Handler (MSH)
1297 about a message previously sent
- 1298 • the Message Service Handler that receives the request sending a Message Status
1299 Response message in return.

1300 9.1.1 Message Status Request Message

1301 A Message Status Request message consists of no *ebXML Payload* and the following elements
1302 in the ebXML Header:

- 1303 • A **Header** element
- 1304 • A **RoutingHeaderList** element
- 1305 • A **Signature** element

1306 The **RoutingHeaderList** and the **Signature** elements MAY be omitted (see sections 8.5 and
1307 8.10).

1308 The **Header** element MUST contain the following:

- 1309 • a **From** element that identifies the party that created the message status request
1310 message
- 1311 • a **To** element that identifies a Party that should receive the message. If a **RoutingHeader**
1312 was present on the message whose status is being checked then this MUST be the
1313 **ReceiverURI** from that message.
- 1314 • a **Service** element that contains:
1315 **<http://www.ebxml.org/namespaces/messageService/MessageStatus>**
- 1316 • an **Action** element that contains **Request**

1317 The message is then sent to the *To Party*.

1318 9.1.2 Message Status Response Message

1319 Once the To Party on the Message Status Request message receives the message, they MAY
1320 generate a Message Status Response message that consists of no ebXML Payload and the
1321 following elements in the ebXML Header.

- 1322 • a **Header** element
- 1323 • a **RoutingHeaderList** element
- 1324 • an **Acknowledgment** element
- 1325 • a **StatusData** element
- 1326 • a **Signature** element

1327 The **RoutingHeaderList**, **Acknowledgment** and **Signature** elements MAY be omitted (see
1328 sections 8.5, 8.9 and 8.10).

1329 The **Header** element MUST contain the following:

- 1330 • a **From** element that identifies the creator of the Message Status Response message
- 1331 • a **To** element that is set to the value of the **From** element in the Message Status Request
1332 message
- 1333 • a **Service** element that contains:
1334 **<http://www.ebxml.org/namespaces/messageService/MessageStatus>**
- 1335 • an **Action** element that contains **Response**
- 1336 • a **RefToMessageId** that identifies the Message Status Request message.

1337 The message is then sent to the *To Party*.

1338 9.1.3 Security Considerations

1339 Party's that receive a Message Status Request message SHOULD always respond to the
1340 message. However they MAY ignore the message instead of responding with **messageStatus**
1341 set to **Unauthorized** if they consider that the sender of the message received is unauthorized.
1342 The decision process that results in this course of action is implementation dependent.

1343 *<DB> Do we want to allow the Message Status Response to include the original response to the
1344 message in the Payload?</DB><CF> quite possibly.</CF>*

1345 9.2 Message Service Handler Ping Service

1346 The Message Service Handler Ping Service enables one Message Service Handler to determine
1347 if another MSH is operating. It consists of:

- 1348 • sending a Message Service Handler Ping message to a MSH, and
- 1349 • the MSH that receives the Ping responding with a Message Service Handler Pong
1350 message.

1351 9.2.1 Message Service Handler Ping Message

1352 A Message Service Handler Ping (MSH Ping) message consists of no ebXML Payload and the
1353 following elements in the ebXML Header:

- 1354 • A **Header** element
- 1355 • A **RoutingHeaderList** element
- 1356 • A **Signature** element

1357 The **RoutingHeaderList** and the **Signature** elements MAY be omitted (see sections 8.5 and
1358 8.10).

1359 The **Header** element MUST contain the following:

- 1360 • a **From** element that identifies the creator of the MSH Ping message
- 1361 • a **To** element that identifies the operator of the MSH that is being sent the MSH Ping
1362 message
- 1363 • a **Service** element that contains:
1364 **<http://www.ebxml.org/namespaces/messageService/MSHStatus>**
- 1365 • an **Action** element that contains **Ping**

1366 The message is then sent to the *To Party*.

1367 9.2.2 Message Service Handler Pong Message

1368 Once the *To Party* on the MSH Ping message receives the message, they MAY generate a
1369 Message Service Handler Pong (MSH Pong) message that consists of no ebXML Payload and
1370 the following elements in the ebXML Header.

- 1371 • a **Header** element
- 1372 • a **RoutingHeaderList** element
- 1373 • an **Acknowledgment** element
- 1374 • a **Signature** element

1375 The **RoutingHeaderList**, **Acknowledgment** and **Signature** elements MAY be omitted (see
1376 sections 8.5, 8.9 and 8.10).

1377 The **Header** element MUST contain the following:

- 1378 • a **From** element that identifies the creator of the MSH Pong message
- 1379 • a **To** element that identifies a Party that generated the MSH Ping message
- 1380 • a **Service** element that contains:
1381 ***http://www.ebxml.org/namespaces/messageService/MessageStatus***
- 1382 • an **Action** element that contains **Pong**
- 1383 • a **RefToMessageId** that identifies the MSH Ping message.

1384 The message is then sent to the To Party.

1385 9.2.3 Security Considerations

1386 Party's that receive a MSH Ping message SHOULD always respond to the message. However
1387 there is a risk that some Parties might use the MSH Ping message to determine the existence of
1388 a Message Service Handler as part of a security attack on that MSH. Therefore recipients of a
1389 MSH Ping MAY ignore the message if they consider that the sender of the message received is
1390 unauthorized or part of some attack. The decision process that results in this course of action is
1391 implementation dependent.

1392 10 Reliable Messaging

1393 *[The Reliable Messaging section has not been agreed to by the membership of the TRP*
1394 *Project Team; however, it is being included to provide a basis for POC developers of MSH*
1395 *implementations. Implementers MUST be prepared for some change to the content of this*
1396 *section.]*

1397 Reliable Messaging defines an interoperable protocol such that the two Messaging Service
1398 Handlers (MSH) operated by a *From Party* and a *To Party* can “reliably” exchange messages that
1399 are sent using “reliable messaging” semantics.

1400 “Reliably” means that the *From Party* can be highly certain that the message sent will be
1401 delivered to the *To Party*. If there is a problem in sending a message then the sender resends the
1402 message until either the message is delivered, or the sender gives up. If the message cannot be
1403 delivered, for example because there has been a catastrophic failure of the *To Party*’s system,
1404 then the *From Party* is informed.

1405 10.1.1 Persistent Storage and System Failure

1406 A MSH that supports Reliable Messaging MUST keep messages that are sent or received reliably
1407 in *persistent storage*. In this context *persistent storage* is a method of storing data that does not
1408 lose information after a system failure or interruption.

1409 This specification recognizes that different degrees of resilience may be realized depending on
1410 the technology that is used to persist the data. However, as a minimum, persistent storage that
1411 has the resilience characteristics of a hard disk (or equivalent) SHOULD be used. It is strongly
1412 RECOMMENDED though that implementers of this specification use technology that is resilient to
1413 the failure of any single hardware or software component.

1414 Even after a system interruption or failure, a MSH MUST ensure that messages in persistent
1415 storage are processed in the same way as if the system failure or interruption had not occurred.
1416 How this is done is an implementation decision.

1417 10.1.2 Methods of Implementing Reliable Messaging

1418 Support for Reliable Messaging can be implemented in one of the following two ways:

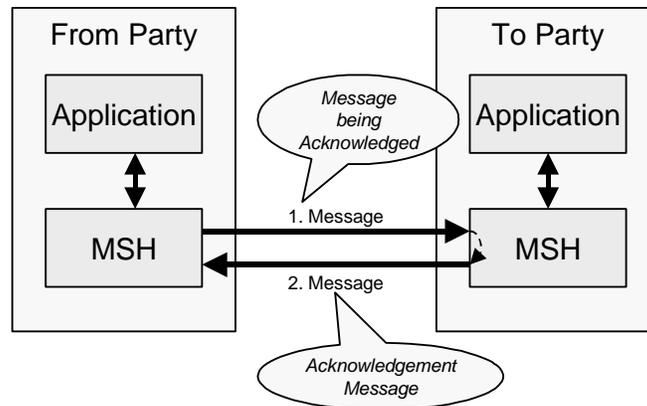
- 1419 • using the ebXML Reliable Messaging protocol, or
- 1420 • using ebXML Header and Message structures together with commercial software
1421 products that are designed to provide reliable delivery of messages using alternative
1422 protocols. <DB>Change elsewhere</DB>

1423 Each of these are described below.

1424 10.2 ebXML Reliable Messaging Protocol

1425 The ebXML Reliable Messaging Protocol described in this section MUST be followed if the
1426 **deliverySemantics** parameter/element is set to **OnceAndOnlyOnce** and the
1427 **ReliableMessagingMethod** parameter/element is set to **ebXML** (the default).

1428 The ebXML Reliable Messaging Protocol is illustrated by the figure below.



1429

1430 **Figure 10-1 Indicating that a message has been received**

1431 The diagram above illustrates two terms that are used in the remainder of this section:

- 1432
- 1433 • *message being acknowledged*. This is the Message that needs to be sent reliably and therefore needs to be acknowledged
 - 1434 • *acknowledgment message*. This is the message that acknowledges that the message being acknowledged has been received.
- 1435

1436 The receipt of the *acknowledgment message* indicates that the *message being acknowledged*
 1437 has been sent reliably.

1438 An *acknowledgment message* MUST contain a **MessageData** element with a **RefToMessageId**
 1439 that contains the same value as the **MessageId** element in the *message being acknowledged*.

1440 A Message can be sent reliably either over:

- 1441
- 1442 • a Single-hop i.e. the sending of a message directly from the *From Party's* MSH to the *To Party's* MSH without passing through any intermediate MSHs.
 - 1443 • Multi-hops i.e. the sending of a message indirectly from the *From Party's* MSH to the *To Party's* MSH via one or more intermediate MSHs.
- 1444

1445 Single-hop Reliable Messaging is described first followed by Multi-hop Reliable Messaging. Note
 1446 that Multi-hop Reliable Messaging is an extension of Single-hop reliable Messaging.

1447 **10.2.1 Single-hop Reliable Messaging**

1448 This section describes the REQUIRED behavior of a Message Service Handler (MSH) that is
 1449 sending and/or receiving messages that support the ebXML Reliable Messaging Protocol.

1450 **10.2.1.1 Sending Message Behavior**

1451 If a MSH is given data by an application that needs to be sent reliably then the MSH MUST do the
 1452 following:

1453 ~~4~~1) Create a message from components received from the application that includes:

- 1454 a) **deliverySemantics** set to **OnceAndOnlyOnce**, and
- 1455 b) a **RoutingHeader** element that identifies the sender and the receiver URIs

1456 ~~5~~2) Save the message in *persistent storage* (see section 10.1.1)

1457 ~~6~~3) Send the message (the *message being acknowledged*) to the *Receiver* MSH

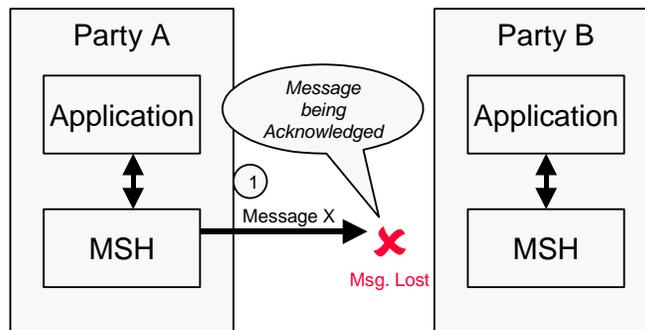
1458 ~~7~~4) Wait for the *Receiver* MSH to return an *acknowledgment message* and, if it does not, then
 1459 resend the *identical* message as described in section 10.2.1.3

- 1460 It is RECOMMENDED that messages that are sent reliably include **deliveryReceiptRequested**
 1461 set to **Signed** or **Unsigned**.
- 1462 If the message does not need to be sent reliably, then **deliverySemantics** MUST be set to
 1463 **BestEffort** (the default).
- 1464 **10.2.1.2 Receiving Message Behavior**
- 1465 If **deliverySemantics** on the received message is set to **OnceAndOnlyOnce** then do the
 1466 following:
- 1467 1) Check to see if the message is a duplicate (e.g. there is a message in *persistent storage* that
 1468 was received earlier that contains the same value for the **MessageId**)
 - 1469 2) If the message is not a duplicate then do the following:
 - 1470 a) Save the **MessageId** of the received message in *persistent storage*. As an
 1471 implementation decision, the whole message MAY be stored if there are other reasons
 1472 for doing so. <DB>Need to re-look at how duplicates are detected if sequence numbers
 1473 are used. </DB>
 - 1474 b) If the received message contains a **RefToMessageId** element then do the following:
 - 1475 i) Look for a message in *persistent storage* that has a **MessageId** that is the same as
 1476 the value of **RefToMessageId** on the received Message
 - 1477 ii) If a message is found in *persistent storage* then mark the persisted message as
 1478 delivered
 - 1479 c) If **deliveryReceiptRequested** is set to **Signed** or **Unsigned** then create an
 1480 **Acknowledgment** element with **type** set to **DeliveryReceipt** that identifies the *received*
 1481 *message*
 - 1482 d) If **syncReplyMode** is set to **True** then pass the data in the received message to the
 1483 application or other process that needs to process it and wait for the application to
 1484 produce a response.
 - 1485 e) If **deliveryReceiptRequested** is set to **Signed** or **Unsigned**, or **syncReplyMode** is set
 1486 to **True** then do the following:
 - 1487 i) Create a **RoutingHeader** element that identifies the sender and the receiver URIs
 - 1488 ii) Set the **RefToMessageId** to the value of the **MessageId** in the received message
 - 1489 iii) Create a *message* from the response generated by the application (if any), the
 1490 **Acknowledgment** element (if any) and the **RoutingHeader** that includes
 1491 **deliverySemantics** set to **OnceAndOnlyOnce**
 - 1492 iv) Save the message in *persistent storage* for later resending
 - 1493 v) Send the message back to the Sending MSH
 - 1494 f) If **syncReplyMode** is set to **False** then pass the data in the received message to the
 1495 application or other process that needs to process it. Note that, depending on the
 1496 application, this can result in the application generating another message to be sent (see
 1497 previous section).
 - 1498 3) If the message is a duplicate, then do the following:
 - 1499 a) Look in *persistent storage* for a response to the received message (i.e. it contains a
 1500 **RefToMessageId** that matches the **MessageId** of the received message) that was *most*
 1501 *recently sent* to the MSH that sent the received message (i.e. it has a **RoutingHeader**
 1502 element with the greatest value of the **Timestamp**)

- 1503 b) If no message was found in *persistent storage* then ignore the received message as
- 1504 either no message was generated in response to the message, or the processing of the
- 1505 earlier message is not yet complete
- 1506 c) If a message was found in *persistent storage* then resend the persisted message back to
- 1507 the MSH that sent the received message.

1508 **10.2.1.3 Resending Lost Messages and Duplicate Filtering**

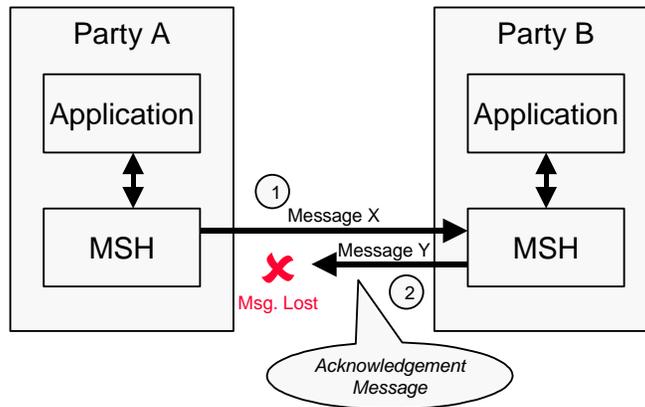
1509 This section describes the behavior that is required by the sender and receiver of a message in
 1510 order to handle when messages are lost. A message is "lost" when a sending MSH does not
 1511 receive a response to a message. For example, it is possible that a *message being*
 1512 *acknowledged* was lost, for example:



1513

1514 **Figure 10-2 Lost "Message Being Acknowledged"**

1515 It is also possible that the *Acknowledgment Message* was lost, for example ...



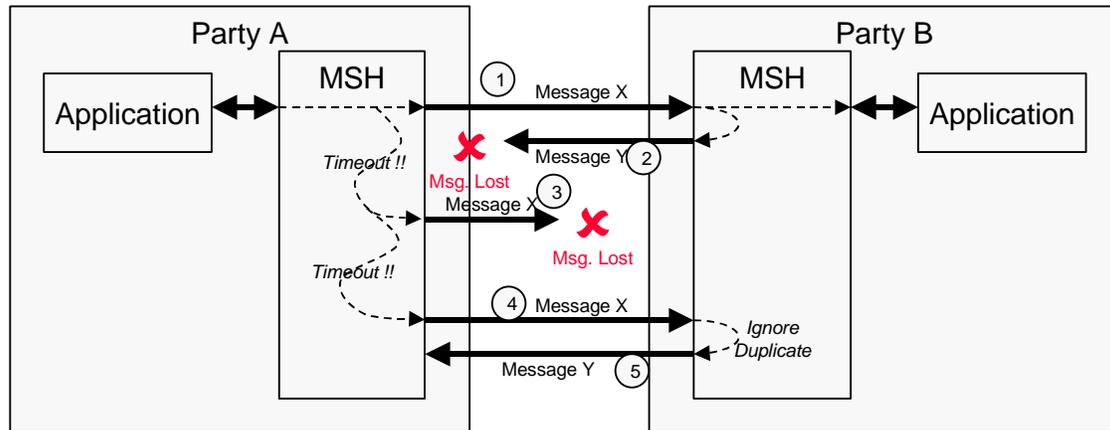
1516

1517 **Figure 10-3 Lost Acknowledgment Message**

1518 The rules that apply are as follows:

- 1519 1) The Sending MSH MUST resend the original message if an *Acknowledgment Message* has
- 1520 not been received from the Receiving MSH and either of the following are true:
 - 1521 a) The message has not yet been resent and at least the time specified in the **timeout**
 - 1522 parameter has passed since the first message was sent, or
 - 1523 b) The message has been resent, and the following are both true:
 - 1524 i) At least the time specified in the **retryInterval** has passed since the last time the
 - 1525 message was resent, and

- 1526 ii) The message has been resent less than the number of times specified in the **retries**
 1527 Parameter
- 1528 2) If the Sending MSH does not receive an *Acknowledgment Message* after the maximum
 1529 number of retries, the Sending MSH SHOULD notify the application and/or system
 1530 administrator function.
- 1531 3) If the Sending MSH detects a communications protocol error that is unrecoverable at the
 1532 transport protocol level then the Sending MSH SHOULD first attempt to resend the message
 1533 using the same transport protocol until the number of **retries** has been reached, and then
 1534 again, using a different communications protocol, if the CPA allows this. If these are not
 1535 successful, then notify the From Party of the failure to deliver as described in section 10.5.



1536

1537 **Figure 10-4 Resending Lost Messages**

1538 The diagram above shows the behavior that MUST be followed by the sender of the *message*
 1539 *being acknowledged* (e.g. Message X) and the *acknowledgment message* (e.g. Message Y).
 1540 Specifically:

- 1541 1) The sender of the *message being acknowledged* (e.g. Party A) MUST re-send the *identical*
 1542 *message* to the *To Party MSH* (e.g. Party B) if no *Acknowledgment Message* is received
- 1543 2) The recipient of the *message being acknowledged* (e.g. Party B), when it receives a *duplicate*
 1544 *message*, MUST re-send to the sender of the *message being acknowledged* (e.g. Party A), a
 1545 message identical to the *most recent message* that was sent to the recipient (i.e. Party A)
- 1546 3) The recipient of the *message being acknowledged* (e.g. Party A) MUST ignore *duplicate*
 1547 *messages* and not forward them a second time to the application, the next MSH *<DB>next*
 1548 *MSH is multi-hop, should not be here. </DB>* or other process that ultimately needs to receive
 1549 them.

1550 *<DB>The above also includes recipient behavior which is not part of sending behavior. Should be*
 1551 *in a separate section. </DB>*

1552 In this context:

- 1553 • an *identical message* is a *message* that contains, apart from perhaps an additional
 1554 **RoutingHeader** element, the same *ebXML Header* and *ebXML Payload* as the earlier
 1555 *message* that was sent.
- 1556 • a *duplicate message* is a *message* that contains the same **MessageId** as an earlier
 1557 *message* that was received.
- 1558 • the *most recent message* is the message with the latest **Timestamp** in the **MessageData**
 1559 element that has the same **RefToMessageId** as the duplicate message that has just
 1560 been received. *<DB>Chris Ferris, disagrees with resending the latest message. DB & CF*
 1561 *need to go through this. </DB>*

1562 Note that the Communication Protocol Envelope MAY be different. This means that the same
 1563 message MAY be sent using different communication protocols and the reliable messaging
 1564 behavior described in this section will still apply. The ability to use alternative communication
 1565 protocols is specified in the CPA.

1566 **10.2.2 Multi-hop Reliable Messaging**

1567 Multi-hop reliable Messaging can occur either:

- 1568 • without Intermediate Acknowledgment, or
- 1569 • with Intermediate Acknowledgments

1570 One reason for using Multi-hop Reliable Messaging with Intermediate Acknowledgments is when
 1571 the *From Party* that is sending a message is confident that the total time taken for ...

- 1572 • the *message being acknowledged* to be sent to the *To Party*, and
- 1573 • the *acknowledgment message* to be returned

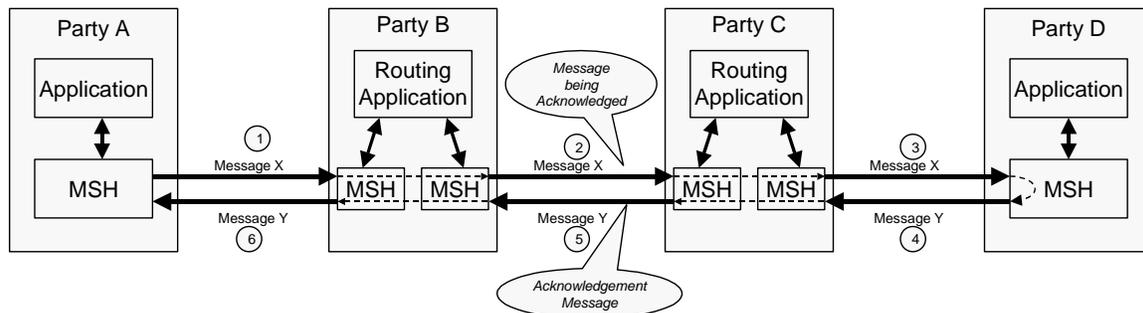
1574 ... is likely to result in the *From Party* resending the *message being acknowledged*. <DB>Chris
 1575 thinks this is superfluous, David thinks it useful as it explains why you should do multi-hop and
 1576 helps an implementer decide when to use it. This requires further discussion. </DB>

1577 Each of these is described below.

1578 **10.2.2.1 Multi-hop Reliable Messaging without Intermediate Acknowledgments**

1579 Multi-hop Reliable Messaging without Intermediate Acknowledgment is identified by the
 1580 **IntermediateAckRequested** of the *Routing Header* for the hop being set to **False** (the default).

1581 The overall message flow is illustrated by the diagram below.



1582

1583 **Figure 10-5 Multi-hop Reliable Messaging without Intermediate Acknowledgments**

1584 This is essentially the same as Single-hop Reliable Messaging except that the Message passes
 1585 through multiple intermediate parties. This means that:

- 1586 • the *From Party* (e.g. Party A) and the *To Party* (e.g. Party D) are the only parties that
 1587 adopt the Reliable Messaging behavior described in this section
- 1588 • the intermediate parties (e.g. Parties B and C), just forward the messages they receive,
 1589 they do not undertake any Reliable Messaging behavior.

1590 This is described in more detail below:

- 1591 1) The *From Party* and the *To Party* adopt the sending message and receiving message
 1592 behavior described in sections 10.2.1.1 and 10.2.1.2 except that the *From Party* MSH (e.g.
 1593 Party A) sends to an Intermediate Party (e.g. Party B) a message (the *message being*
 1594 *acknowledged*) e.g. Message X in transmission 1, that contains

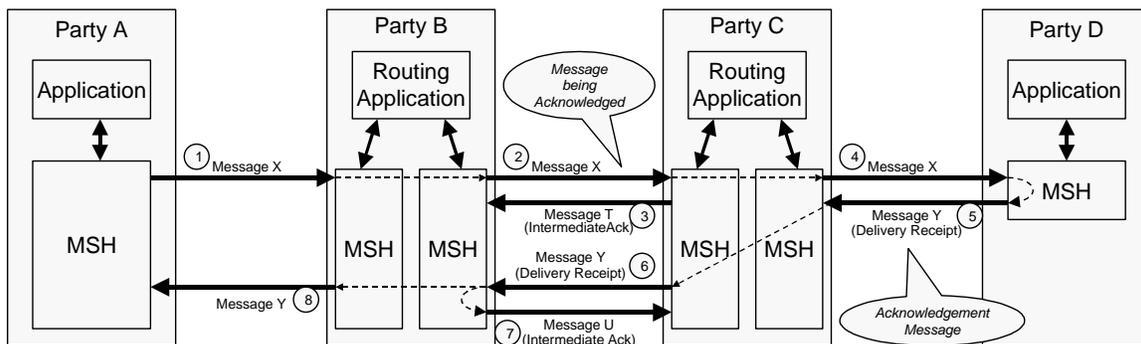
- a) a **QualityOfServiceInfo** element with **deliverySemantics** set to **OnceAndOnlyOnce**

- 1596 b) a **RoutingHeader** element that contains the **SenderURI** of the sender (e.g. the URL for
 1597 Party A's MSH) and the **ReceiverURI** of the next recipient of the message (e.g. the URL
 1598 of Party B's MSH)
- 1599 2) Once the Intermediate Party (e.g. Party B or Party C) receives the message, they determine
 1600 its next destination (in the example above this could be done by the Routing Application) and
 1601 forward the message (e.g. Transmission 2 of Message X) to the next Party (e.g. either Party
 1602 C or Party D). Before sending the message they do the following:
- 1603 a) transfer elements in the ebXML Header and Payload unchanged from the inbound
 1604 message to the outbound message except that, they
- 1605 b) add a **RoutingHeader** element to the **RoutingHeaderList** that contains the **SenderURI**
 1606 of the next party to receive the message (e.g. the URL for Party C's or Party D's MSH)
 1607 and the **ReceiverURI** (e.g. the URL for Party B's or Party C's MSH)
- 1608 3) If the Sending MSH (either at the From Party or at an Intermediate Party) does not receive an
 1609 *Acknowledgment Message* after the maximum number of retries, the Sending MSH SHOULD
 1610 notify the following of the delivery failure:
- 1611 a) The application and/or system administrator function if the Sending MSH is the *From*
 1612 Party MSH, or
- 1613 b) The Sending MSH of the *From Party*, if the Sending MSH is operated by an Intermediate
 1614 Party (see section 10.5)
- 1615 4) The previous step then repeats until eventually the message (e.g. Message X) reaches its
 1616 final destination at the *To Party* (e.g. Party D)
- 1617 5) Once the *To Party* receives the message (i.e. the *message being acknowledged*) they return
 1618 an *acknowledgment message* to the *From Party* through the Intermediate Parties.)
- 1619 6) Steps 2 and 3 above then repeat until the *acknowledgment message* reaches the *To Party*
 1620 (e.g. Party A)

1621 **10.2.2.2 Multi-hop Reliable Messaging with Intermediate Acknowledgments**

1622 Multi-hop Reliable Messaging with Intermediate Acknowledgments is similar to Multi-hop Reliable
 1623 Messaging without Intermediate Acknowledgment except that any of the Parties that are
 1624 transmitting a Message can request that the recipient return an *Intermediate Acknowledgment*.

1625 This is illustrated by the diagram below.



1626

1627 **Figure 10-6 Multi-hop Reliable Messaging with Intermediate Acknowledgments**

1628 The main difference between Multi-Hop Reliable Messaging with Intermediate Acknowledgments
 1629 and the without is:

- 1630 • any party may request an intermediate acknowledgment

- 1631 • any party that either sends or receives a message that requests an intermediate
 1632 acknowledgment must adopt the reliable messaging behavior even if the
 1633 **QualityOfServiceInfo** element indicates otherwise.
- 1634 The rules that apply to Multi-hop Reliable Messaging with Intermediate Acknowledgment are as
 1635 follows:
- 1636 1) Any Party that is sending a message can request that the recipient send an *Acknowledgment*
 1637 *Message* that is an *Intermediate Acknowledgment* by setting the
 1638 **IntermediateAckRequested** of the **RoutingHeader** for the hop to **Signed** or **Unsigned**.
 1639 (e.g. Transmission 2 of Message X, or Transmission 6 of Message Y)
- 1640 2) If a MSH that is not the *To Party* receives a message that requires an Intermediate
 1641 Acknowledgment (e.g. Transmission 2 of Message X, or Transmission 6 of Message Y) then:
- 1642 a) If the MSH can identify itself as the **ReceiverURI** in the **RoutingHeader** for the hop, and
 1643 an *Intermediate Acknowledgment* is requested, then the MSH must return an
 1644 *Acknowledgment Message* (e.g. Transmission 3 of Message T, or Transmission 7 of
 1645 Message U) with:
- 1646 i) The **Service** and **Action** elements set as in defined in section 10.4
- 1647 ii) The **From** element contains the **ReceiverURI** from the last **RoutingHeader** in the
 1648 message that has just been received
- 1649 iii) The **To** element contains the **SenderURI** from the last **RoutingHeader** in the
 1650 message that has just been received
- 1651 iv) a **RefToMessageld** element that contains the **Messageld** of the message being
 1652 acknowledged
- 1653 v) a **QualityOfServiceInfo** element with **deliverySemantics** set to
 1654 **OnceAndOnlyOnce**
- 1655 vi) an **Acknowledgment** element with type set to **IntermediateAck**
- 1656 vii) a **RoutingHeader** element that contains the **SenderURI** of the sender (e.g. the URL
 1657 for Party C's or Party B's MSH) and the **ReceiverURI** of the next recipient of the
 1658 message (e.g. the URL of Party B's or Party C's MSH)
- 1659 3) If a MSH that is the *To Party* receives a message and it requires an Intermediate
 1660 Acknowledgment (see step 2) then, unless the *To Party* is returning an *Acknowledgment*
 1661 *Message* that is a *Delivery Receipt*, return an *Acknowledgment Message* as described in step
 1662 2c above.

1663 10.3 ebXML Reliable Messaging using Queuing Transports

1664 This section describes the differences that apply if a Queuing Transport is used to implement
 1665 Reliable Messaging.

1666 Use of the ebXML Reliable Messaging Protocol is identified by the **ReliableMessagingMethod**
 1667 parameter being set to **Transport** for transmission (either a Single-hop or a Multi-hop)

1668 If Reliable Messaging using a Queuing Transport is being used then the following rules apply:

- 1669 1) An Intermediate Ack SHOULD not be requested. If an Intermediate Ack is requested, then it
 1670 is ignored.
- 1671 2) No message acknowledgments with an **Acknowledgment** element with a **type** of
 1672 **IntermediateAck** should be sent, even if requested
- 1673 3) Implementations should use the facilities of the Queuing Transport to determine if the
 1674 message was delivered

- 1675 4) If an intermediate MSH cannot forward a message to the next Party then the From Party
 1676 should be notified using the procedure described in section 10.5.
- 1677 5) An acknowledgment message with an **Acknowledgment** element with a type attribute set to
 1678 **deliveryReceipt** can be sent if requested to inform the sender of the message being
 1679 acknowledged that the message was delivered.

1680 **10.4 Service and Action Element Values**

1681 An **Acknowledgment** element can be included in an **ebXMLHeader** that is part of a *message*
 1682 that is being sent as a result of processing of an earlier message. In this case the values for the
 1683 **Service** and **Action** elements are set by the designer of the Service (see section 8.4.4).

1684 An **Acknowledgment** element also can be included in an **ebXMLHeader** that does not include
 1685 any results from the processing of an earlier message. In this case, the values of the **Service** and
 1686 **Action** elements MUST be set as follows:

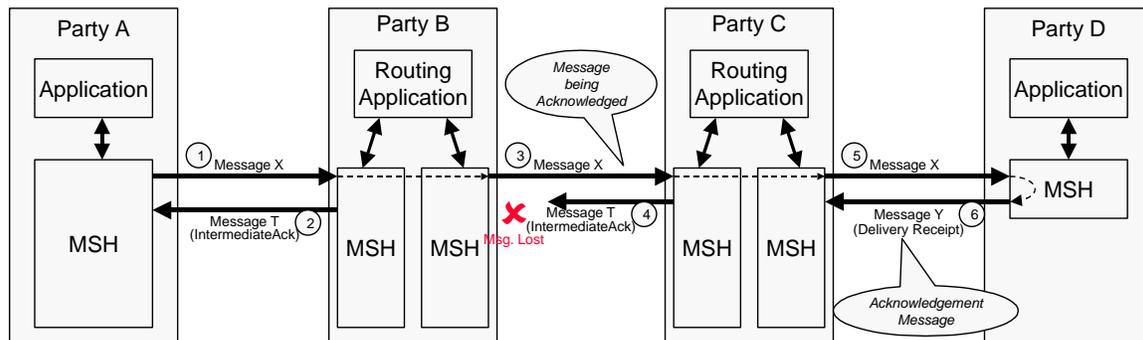
- 1687 • The **Service** element MUST be set to:
 1688 ***http://www.ebxml.org/namespaces/messageService/MessageAcknowledgment***
- 1689 • The **Action** element MUST be set to the value of the **type** attribute in the
 1690 **Acknowledgment** element.

1691 Note that **deliveryReceiptRequested** must be set to **None** on a message that is only an
 1692 acknowledgment.

1693 **10.5 Failed Message Delivery**

1694 It is possible, that a Message cannot be delivered to its ultimate destination. This can be either:

- 1695 • when the *To Party* MSH cannot deliver the message to the Application or other process
 1696 that needs it, or
- 1697 • when using Intermediate Acknowledgments and an Intermediate system determines that
 1698 a message may have been lost. This is illustrated by the diagram below.



1699

1700 **Figure 10-7 Failed Message Delivery using Intermediate Acknowledgments**

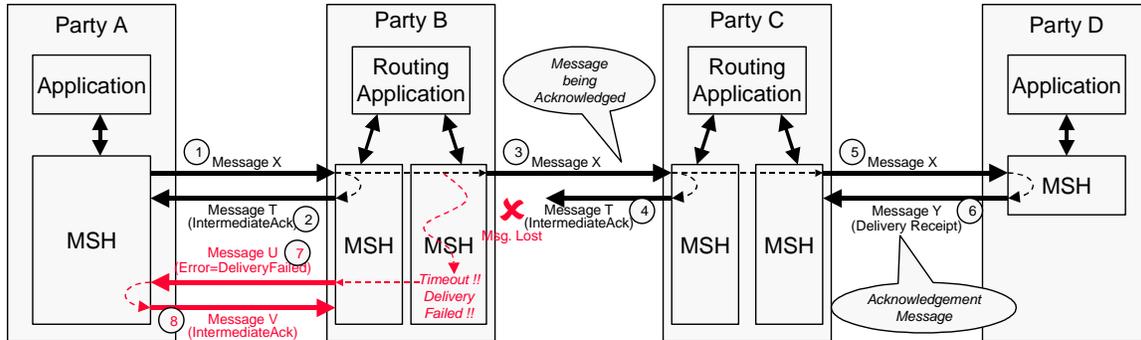
1701 In this example, Party B does not know if Party C (or Party D) has received the message since,
 1702 even after resending, it has not received the *acknowledgment message* (Message T).

1703 In both these circumstances the MSH that detects the problem MUST send a message to the
 1704 *From Party* that sent the *message being acknowledged* (via the Intermediate Party if required).
 1705 The message contains:

- 1706 • a **From Party** that identifies the Party that detected the problem
- 1707 • a **To Party** that identifies the **From Party** that created the message that could not be
 1708 delivered
- 1709 • a **Service** element and **Action** element set as described in 11.5

- 1710 • a **QualityOfServiceInfo** element with **deliverySemantics** set to the same value as the
- 1711 **deliverySemantics** on the message that could not be delivered
- 1712 • an **Error** element with a severity of:
 - 1713 - **Error** if the Party that detected the problem could not even transmit the message
 - 1714 (e.g. Transmission 3 was impossible)
 - 1715 - **Warning** if the message (e.g. Message X in Transmission 3) was transmitted, but no
 - 1716 acknowledgment was received. This means that the message probably was not
 - 1717 delivered although there is a small probability that it was
- 1718 • an **ErrorCode** of **DeliveryFailure**

1719 This is illustrated by the diagram below by the text and arrows in red.



1720

1721 **Figure 10-8 Reporting Failed Message Delivery**

1722 Note that the message that contains an **Error** element with an **ErrorCode** of **DeliveryFailure**

1723 (e.g. Message U in Transmission 7) might be sent reliably. It is possible the *acknowledgment*

1724 *message* for this message (e.g. Message V in Transmission 8) is not received. In this case, the

1725 Party that detects the failed delivery (e.g. Party B) SHOULD inform the Party (e.g. Party A) that

1726 sent the *message being acknowledged* (e.g. Message X in Transmission 1) of the failure. How

1727 this is done is outside the scope of this specification.

1728 **10.6 Reliable Messaging Parameters**

1729 This section describes the parameters required to control reliable messaging. This parameter

1730 information may be contained:

- 1731 • in the ebXML Message header, or
- 1732 • in the CPA associated with the message.

1733 If the information is in both the ebXML message header and the CPA, the information in the

1734 header over-rides the CPA.

1735 **10.6.1 Who sets Message Service Parameters**

1736 The values to be used in parameters can be specified by the following parties:

- 1737 • the *From Party*
- 1738 • the *To Party*
- 1739 • the sending Message Service Handler (MSH)
- 1740 • the receiving Message Service Handler

1741 Parameters set by the *From Party* or the *To Party*, apply to the delivery of a message as a whole.

1742 Parameters set by the sending or receiving MSH apply to a single-hop.

1743 Note that the *From Party* is the sending MSH and the *To Party* is the receiving MSH for the
 1744 first/last MSH that handles the message.

1745 The table below indicates where these parameters may be set.

1746

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
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

1747 In this table, the following interpretation of the columns should be used:

1748 7) the **Specified By** columns indicates the Party that sets the value in the Collaboration Party
 1749 Protocol, Message Header, or Routing Header

1750 8) if the **CPA/ CPP** column contains a **Yes** then it indicates that the party in the **Specified By**
 1751 column specifies the value that is present in the **CPP**

1752 9) if the **CPA/ CPP** column contains a **No** then it indicates that the parameter value is never
 1753 specified in the **CPP**

1754 10) if the **Message Header** or **Routing Header** columns contain a **Yes** then it indicates that the
 1755 parameter value may be specified in the **Header** element or **Routing Header** and over-rides
 1756 any value in the **CPA**. If the value is not specified in the **Header element** or **Routing Header**
 1757 then the value in the **CPA** must be used.

1758 11) if the **Message Header/Routing Header** columns contain a **No** then it indicates that the
 1759 value in the **CPA** is always used

1760 12) if the **Message Header/Routing Header** columns contain a **N/A** then it indicates that the
 1761 value may be specified in another header

1762 These parameters are described below.

1763 10.6.2 From Party Parameters

1764 This section describes the parameters that are set by the *From Party*

1765 10.6.2.1 Delivery Semantics

1766 The *deliverySemantics* parameter may be present as either an element within the
 1767 *ebXMLHeader* element or as a parameter within the CPA. See section 8.4.7.1 for more
 1768 information.

1769 10.6.2.2 Delivery Receipt Requested

1770 The **deliveryReceiptRequested** parameter may be present as either an element within the
1771 **ebXMLHeader** element or as a parameter within the CPA. See section 8.4.7.2 for more
1772 information.

1773 10.6.2.3 Sync Reply Mode

1774 The **syncReplyMode** parameter may be present as either an element within the **ebXMLHeader**
1775 element or as a parameter within the CPA. See section 8.4.7.3 for more information.

1776 10.6.2.4 Time To Live

1777 The **TimeToLive** element may be presented within the **ebXMLHeader** element see section
1778 8.4.6.4 ~~8.4.7.2~~ for more information.

1779 10.6.3 To Party Parameters

1780 This section describes the parameters that are set by the *To Party*

1781 10.6.3.1 Delivery Receipt Provided

1782 The **DeliveryReceiptProvided** parameter indicates whether a *To Party* can provide an
1783 *acknowledgment message* with a **type** attribute of **deliveryReceipt** in response to a message.
1784 Valid values are:

- 1785 • **Signed** - indicates that only a signed Delivery Receipt can be provided
- 1786 • **Unsigned** - indicates only an unsigned Delivery Receipt can be provided,
- 1787 • **Both** - indicates that either a signed or an unsigned Delivery Receipt can be provided, or
- 1788 • **None** - indicates that the *To Party* does not create Delivery Receipts

1789 If a MSH receives a Message where **deliveryReceiptRequested** is in not compatible with the
1790 value of **DeliveryReceiptProvided** then the MSH MUST return an *Error Message* to the *From*
1791 *Party* MSH, reporting that the **DeliveryReceiptProvided** is not supported. This must contain an
1792 **errorCode** set to **NotSupported** and a **severity** of Error.

1793 10.6.4 Sending MSH Parameters

1794 This section describes the parameters that are set by the *Party* that operates the Sending MSH.

1795 10.6.4.1 Reliable Messaging Method

1796 The **ReliableMessagingMethod** parameter indicates the requested method for Reliable
1797 Messaging that will be used when sending a Message. Valid values are:

- 1798 • **ebXML** in this case the ebXML Reliable Messaging Protocol as defined in section 10.2 is
1799 followed, or
- 1800 • **Transport**, in this case a Queuing Transport Protocol is used for reliable delivery of the
1801 message, see section 10.3.

1802 10.6.4.2 Intermediate Ack Requested

1803 The **IntermediateAckRequested** parameter is used by the Sending MSH to request that the
1804 Receiving MSH that receives the *Message* returns an *acknowledgment message* with an
1805 **Acknowledgment** element with a **type** of **IntermediateAcknowledgment**.

1806 Valid values for **IntermediateAckRequested** are:

- 1807 • **Unsigned** - requests that an unsigned Delivery Receipt is requested

- 1808 • **Signed** - requests that a signed Delivery Receipt is requested, or
1809 • **None** - indicates that no Delivery Receipt is requested.
1810 The default value is **None**.

1811 **10.6.4.3 Timeout Parameter**

1812 The **timeout** parameter is an integer value that specifies the time in seconds that the Sending
1813 MSH MUST wait for an *Acknowledgment Message* before first resending a message to the
1814 Receiving MSH.

1815 **10.6.4.4 Retries Parameter**

1816 The **retries** Parameter is an integer value that specifies the maximum number of times the
1817 *message being acknowledged* must be resent to the Receiving MSH using the same
1818 Communications Protocol by the Sending MSH.

1819 **10.6.4.5 RetryInterval Parameter**

1820 The **retryInterval** parameter is an integer value specifying, in seconds, the time the Sending
1821 MSH MUST wait between retries, if an *Acknowledgment Message* is not received.

1822 **10.6.4.6 Deciding when to resend a message**

1823 The Sending MSH MUST resend the original message if an *Acknowledgment Message* has not
1824 been received from the Receiving MSH and either:

- 1825 • the message has not yet been resent and at least the time specified in the **timeout**
1826 parameter has passed since the first message was sent, or
1827 • the message has been resent, and
1828 - at least the time specified in the **retryInterval** has passed since the last time the
1829 message was resent, and
1830 - the message has been resent less than the number of times specified in the **retries**
1831 Parameter, and

1832 If the Sending MSH does not receive an *Acknowledgment Message* after the maximum number
1833 of retries, the Sending MSH SHOULD notify either:

- 1834 • the application and/or system administrator function if the Sending MSH is the *From*
1835 *Party* MSH, or
1836 • send an message reporting the delivery failure, if the Sending MSH is operating by an
1837 Intermediate Party (see section 10.5)

1838 **10.6.5 Receiving MSH Parameters**

1839 This section describes the parameters that are set by the *Party* that operates the Receiving MSH.

1840 **10.6.5.1 Reliable Messaging Methods Supported**

1841 The **reliableMessagingMethodsSupported** parameter is a list of the methods that a MSH uses
1842 to support Reliable Messaging. It must be a URI. The URI for the ebXML Reliable Messaging
1843 Protocol described in section 10.2 is **<http://www.ebxml.org/namespaces/reliableMessaging>**

1844 **10.6.5.2 PersistDuration**

1845 **persistDuration** is the minimum length of time, expressed as a [XMLSchema] timeDuration, that
1846 data from a *Message* that is sent reliably, is kept in *Persistent Storage* by a MSH that receives
1847 that *Message*.

1848 In order to support the filtering of duplicate messages, a Receiving MSH MUST, as a minimum,
1849 save the **MessageId** in *persistent storage*. It is also RECOMMENDED that the following be kept
1850 in *Persistent Storage*:

- 1851 • the complete message, at least until the information in the message has been passed to
1852 the application or other process that needs to process it
- 1853 • the time the message was received, so that the information can be used to generate the
1854 response to a Message Status Request (see section 9.1.1)

1855 ***persistDuration*** is specified in the CPA.

1856 A MSH SHOULD NOT resend a message with the same **MessageId** to a receiving MSH if the
1857 elapsed time indicated by ***persistDuration*** has passed since the message was first sent as the
1858 receiving MSH will probably not treat it as a duplicate.

1859 If a message cannot be sent successfully before ***persistDuration*** has passed, then the MSH
1860 should report a delivery failure (see section 10.5).

1861 Note that implementations may determine that a message is persisted for longer than the time
1862 specified in ***persistDuration***, for example in order to meet legal requirements or the needs of a
1863 business process. This information is recorded separately within the CPA.

1864 In order to ensure that persistence is continuous as the message is passed from the receiving
1865 MSH to the process or application that is to handle it, it is RECOMMENDED that a message is
1866 not removed from *persistent storage* until the MSH knows that the data in the message has been
1867 received by the process/application.

1868 **10.6.5.3 MSH Time Accuracy**

1869 The ***mshTimeAccuracy*** parameter in the CPA indicates the minimum accuracy that a Receiving
1870 MSH keeps the clocks it uses when checking, for example, ***TimeToLive***. It's value is in the format
1871 "mm:ss" which indicates the accuracy in minutes and seconds.

1872 11 Error Reporting and Handling

1873 This section describes how one ebXML Message Service Handler (MSH) reports errors it detects
1874 in an ebXML Message to another MSH.

1875 11.1 Definitions

1876 For clarity two phrases are defined that are used in this section:

- 1877 • *message in error*. A message that contains or causes an error of some kind
- 1878 • *message reporting the error*. A message that contains an ebXML **ErrorList element** that
1879 describes the error(s) found in a *message in error*.

1880 11.2 Types of Errors

1881 One MSH needs to report to another MSH errors in a *message in error* that are associated with:

- 1882 • the structure or content of the *Message Envelope* (e.g. MIME) (see section 7),
- 1883 • the ebXML Message Header document (see section 8),
- 1884 • reliable messaging failures (see section 10), or
- 1885 • security (see section 12).

1886 Unless specified to the contrary, all references to "an error" in the remainder of this specification
1887 imply any or all of the types of errors listed above.

1888 Errors associated with Data Communication protocols are detected and reported using the
1889 standard mechanisms supported by that data communication protocol and are do not use the
1890 error reporting mechanism described here.

1891 11.3 When to generate Error Messages

1892 When an MSH detects an error in a *message in error*, a *message reporting the error* MUST be
1893 generated and delivered to the MSH that sent the *message in error* if:

- 1894 • the Error Reporting Location (see section 11.4) to which the *message reporting the error*
1895 should be sent can be determined, and
- 1896 • the *message in error* does not have an **ErrorList** element with **highestSeverity** set to
1897 **Error**.

1898 If the Error Reporting Location cannot be found or the *message in error* has an **ErrorList** element
1899 with **highestSeverity** set to **Error**, it is RECOMMENDED that:

- 1900 • the error is logged,
- 1901 • the problem is resolved by other means, and
- 1902 • no further action is taken.

1903 11.3.1 Security Considerations

1904 Party's that receive a Message that contains an error in the header SHOULD always respond to
1905 the message. However they MAY ignore the message and not respond if they consider that the
1906 message received is unauthorized or is part of some security attack. The decision process that
1907 results in this course of action is implementation dependent.

1908 11.4 Identifying the Error Reporting Location

1909 The Error Reporting Location is a URI that is specified by the sender of the *message in error* that
1910 indicates where to send a *message reporting the error*. This may be specified:

- 1911 • by reference, for example by using the **CPAId** to identify the Party Agreement that
1912 contains the Error Reporting Location, or
- 1913 • by value, for example by using the **ErrorURI** contained within the **RoutingHeader**
1914 element.
- 1915 If a *message* contains an **ErrorURI** then the **ErrorURI** MUST be used.
- 1916 If an **ErrorURI** is not used then the **ErrorURI** implied by the CPA identified by the **CpaID** on the
1917 message SHOULD be used. If no **ErrorURI** is implied by the CPA, then the **SenderURI** MUST be
1918 used.
- 1919 Even if the *message in error* cannot be successfully analyzed or parsed, MSH implementers
1920 SHOULD try to determine the Error Reporting Location by other means. How this is done is an
1921 implementation decision.

1922 **11.5 Service and Action Element Values**

- 1923 An **ErrorList** element can be included in an **ebXMLHeader** that is part of a *message* that is being
1924 sent as a result of processing of an earlier message. In this case, the values for the **Service** and
1925 **Action** elements are set by the designer of the Service (see section 8.4.4).
- 1926 An **ErrorList** element can also be included in an **ebXMLHeader** that is not being sent as a result
1927 of the processing of an earlier message. In this case, the values of the **Service** and **Action**
1928 elements MUST be set as follows:
- 1929 • The **Service** element MUST be set to:
1930 **<http://www.ebxml.org/namespaces/messageService/MessageStatus>**
- 1931 • The **Action** element MUST be set to **MessageError**.

1932 **12 Security**

1933 The ebXML Message Service, by its very nature, presents certain security risks. A Message
1934 Service may be at risk by means of:

- 1935 • Unauthorized access
- 1936 • Data integrity and/or confidentiality attacks (e.g. through man-in-the-middle attacks)
- 1937 • Denial-of-Service, spoofing, bombing attacks

1938 Each security risk is described in detail in the ebXML Technical Architecture Security
1939 Specification [EBXMLSEC].

1940 Each of these security risks MAY be addressed in whole, or in part, by the application of one, or a
1941 combination, of the countermeasures described in this section. This specification describes a set
1942 of profiles, or combinations of selected countermeasures, that have been selected to address key
1943 risks based upon commonly available technologies. Each of the specified profiles includes a
1944 description of the risks that are not addressed.

1945 Application of countermeasures SHOULD be balanced against an assessment of the inherent
1946 risks and the value of the asset(s) that might be placed at risk.

1947 **12.1 Security and Management**

1948 No technology, regardless of how advanced it might be, is an adequate substitute to the effective
1949 application of security management policies and practices.

1950 It is **STRONGLY RECOMMENDED** that the site manager of an ebXML Message Service apply
1951 due diligence to the support and maintenance of its; security mechanism, site (or physical)
1952 security procedures, cryptographic protocols, update implementations and apply fixes as
1953 appropriate. (See <http://www.cert.org/> and <http://ciac.llnl.gov/>)

1954 **12.2 Collaboration Protocol Agreement**

1955 The configuration of Security for MSHs is specified in the CPA. Three areas of the CPA have
1956 security definitions as follows:

- 1957 • The Document Exchange section addresses security to be applied to the payload of the
1958 message. The MSH is not responsible for any security specified at this level but may
1959 offer these services to the message sender.
- 1960 • The Message section addresses security applied to the entire ebXML Document, which
1961 includes the header and the payload.
- 1962 • The Transport section addresses the Transport level. The MSH is not responsible for
1963 any security specified at this level.

1964 **12.3 Countermeasure Technologies**

1965 **12.3.1 Persistent Digital Signature**

1966 If signatures are being used to digitally sign an ebXML message then XML Signature [DSIG]
1967 MUST be used to bind the ebXML Header Document to the ebXML Payload or data elsewhere on
1968 the web that relates to the message. It is also strongly **RECOMMENDED** that XML Signature is
1969 used to digitally sign the Payload on its own.

1970 The only available technology that can be applied to the purpose of digitally signing an ebXML
1971 Message (both the ebXMLHeader and its associated payload objects) is provided by technology
1972 that conforms to the W3C/IETF joint XML Signature specification [XMLDSIG]. An XML Signature
1973 conforming to this specification can selectively sign portions of an XML document(s), permitting

1974 the documents to be augmented (new element content added) while preserving the validity of the
1975 signature(s).

1976 An ebXML Message that requires a digital signature SHALL be signed following the processed
1977 defined in this section of the specification and SHALL be in full compliance with [XMLDSIG].

1978 **12.3.1.1 Signature Generation**

1979 13) Create a SignedInfo element with SignatureMethod, CanonicalizationMethod, and
1980 Reference(s) elements for the ebXMLHeader document and any required payload objects, as
1981 prescribed by [XMLDSIG].

1982 14) Canonicalize and then calculate the SignatureValue over SignedInfo based on algorithms
1983 specified in SignedInfo as specified in [XMLDSIG].

1984 15) Construct the Signature element that includes the SignedInfo, KeyInfo (RECOMMENDED),
1985 and SignatureValue elements as specified in [XMLDSIG].

1986 16) Include the namespace qualified Signature element in the ebXMLHeader document just
1987 signed, following the RoutingHeaderList element.

1988 The ds:SignedInfo element SHALL be composed of zero or one ds:CanonicalizationMethod
1989 element, the ds:SignatureMethod and one or more ds:Reference elements.

1990 The ds:CanonicalizationMethod element is defined as OPTIONAL in [XMLDSIG], meaning that
1991 the element need not appear in an instance of a ds:SignedInfo element. The default
1992 canonicalization method that is applied to the data to be signed is [XMLC14N] in the absence of a
1993 ds:Canonicalization element that specifies otherwise. This default SHALL also serve as the
1994 default canonicalization method for the ebXML Message Service.

1995 The ds:SignatureMethod element SHALL be present and SHALL have an Algorithm attribute. The
1996 RECOMMENDED value for the Algorithm attribute is:

1997 <http://www.w3.org/2000/02/xmlsig#sha1>

1998 This RECOMMENDED value SHALL be supported by all compliant ebXML Message Service
1999 software implementations.

2000 The ds:Reference element for the ebXMLHeader document SHALL have an URI attribute value
2001 of "" to provide for the signature to be applied to the document that contains the ds:Signature
2002 element (the ebXMLHeader document). The ds:Reference element for the ebXMLHeader
2003 document MAY include a Type attribute that has a value
2004 "http://www.w3.org/2000/02/xmlsig#Object" in accordance with [XMLDSIG]. This attribute is
2005 purely informative. It MAY be omitted. Implementations of the ebXML MSH SHALL be prepared
2006 to handle either case. The ds:Reference element MAY include the optional id attribute.

2007 The ds:Reference element for the ebXMLHeader document SHALL include a child ds:Transform
2008 element that excludes the containing ds:Signature element and all its descendants as well as the
2009 RoutingHeaderList element and all its descendants as these elements are subject to change. The
2010 ds:Transform element SHALL include a child ds:XPath element that has a value of:

2011 `/descendant-or-self::node()[not(ancestor-or-self::ds:Signature[@id='S1']) and not(ancestor-or-`
2012 `self::RoutingHeaderList)]`
2013

2014 Each payload object that requires signing SHALL be represented by a ds:Reference element that
2015 SHALL have an URI attribute that resolves to that payload object. This MAY be either the
2016 Content-Id URI of the payload object enveloped in the MIME ebXML Payload Container, or an
2017 URI that matches the Content-Location header of the payload object enveloped in the ebXML
2018 Payload Container, or an URI that resolves to an external payload object that is external to the
2019 ebXML Payload Container. It is STRONGLY RECOMMENDED that the URI attribute value match
2020 the xlink:href URI value of the corresponding Manifest/Reference element for that payload object.
2021 However, this is NOT REQUIRED.

2022 Example of digitally signed ebXMLHeader document:

2023
2024
2025
2026
2027
2028
2029
2030
2031
2032
2033
2034
2035
2036
2037
2038
2039
2040
2041
2042
2043
2044
2045
2046
2047
2048
2049
2050
2051
2052
2053
2054
2055
2056
2057
2058
2059
2060
2061
2062
2063
2064
2065
2066

```
<?xml version="1.0" encoding="utf-8"?>
<ebXMLHeader
  xmlns="http://www.ebxml.org/namespaces/messageHeader"
  xmlns:xlink="http://www.w3.org/1999/xlink"
  version="1.0">
  <Manifest id="Mani01">
    <Reference xlink:href="cid://blahblahblah"
      xlink:role="http://ebxml.org/gci/invoice">
      <Schema version="1.0" location="http://ebxml.org/gci/busdocs/invoice.dtd"/>
    </Reference>
  </Manifest>
  <Header>
    ...
  </Header>
  <RoutingHeaderList>
    <RoutingHeader>
      ...
    </RoutingHeader>
  </RoutingHeaderList>
  <ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmlds#">
    <ds:SignedInfo>
      <ds:CanonicalizationMethod Algorithm="http://www.w3.org/TR/2000/WD-xml-c14n-20001011"/>
      <ds:SignatureMethod Algorithm="http://www.w3.org/2000/09/xmlds#dsa-shal"/>
      <ds:Reference URI=" ">
        <ds:Transforms>
          <ds:Transform>
            <XPath>/descendant-or-self::node()[not(ancestor-or-self::ds:Signature[@id='S1']) and
not(ancestor-or-self::RoutingHeaderList)]</XPath>
          </ds:Transform>
        </ds:Transforms>
      <ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmlds#shal"/>
      <ds:DigestValue>...</ds:DigestValue>
    </ds:Reference>
    <ds:Reference URI="cid://blahblahblah/">
      <ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmlds#shal"/>
      <ds:DigestValue>...</ds:DigestValue>
    </ds:Reference>
  </ds:SignedInfo>
  <ds:SignatureValue>...</ds:SignatureValue>
  <ds:KeyInfo>...</ds:KeyInfo>
</ds:Signature>
</ebXMLHeader>
```

2067 **12.3.2 Persistent Signed Receipt**

2068 An ebXML Message that has been digitally signed MAY be acknowledged with a DeliveryReceipt
2069 acknowledgment message that itself is digitally signed in the manner described in the previous
2070 section. The acknowledgment message MUST contain the set of ds:DigestValue elements
2071 contained in the ds:Signature element of the original message within the Acknowledgment
2072 element.

2073 **12.3.3 Non-persistent Authentication**

2074 Non-persistent authentication is provided by the communications channel used to transport the
2075 ebXML message. This authentication MAY be either in one direction—from the session initiator to
2076 the receiver—or bi-directional. The specific method will be determined by the communications
2077 protocol used. For instance, the use of a secure network protocol, such as [TLS] or [IPSEC]
2078 provides the sender of an ebXML Message to authenticate the destination for the TCP/IP
2079 environment.

2080 **12.3.4 Non-persistent Integrity**

2081 Use of a secure network protocol such as [TLS] or [IPSEC] MAY be configured so as to provide
2082 for integrity check CRCs of the packets transmitted via the network connection.

2083 12.3.5 Persistent Confidentiality

2084 XML Encryption is ~~an~~ a W3C/IETF joint activity that is actively engaged in the drafting of a
2085 specification for the selective encryption of an XML document(s). It is anticipated that ~~this~~
2086 ~~specification~~this specification will be completed within the next year. The ebXML Transport,
2087 Routing and Packaging team has identified this technology as the only viable means of providing
2088 persistent, selective confidentiality of elements within an ebXML Message including the
2089 ebXMLHeader document.

2090 Confidentiality for ebXML Payloads MAY be provided by functionality possessed by a MSH.
2091 However, this specification states that it is not the responsibility of the MSH to provide security for
2092 the ebXML Payloads. Payload confidentiality MAY be provided by using XML Encryption (when
2093 available) or some other cryptographic process, such as [S/MIME], [S/MIMEV3], or [PGP/MIME],
2094 that is bilaterally agreed upon by the parties involved. Since XML Encryption is not currently
2095 available, it is RECOMMENDED that [S/MIME] encryption methods be used for ebXML Payloads.
2096 The XML Encryption standard SHALL be the default encryption method when XML Encryption
2097 has achieved W3C Recommendation status.

2098 Section xx (TBD) describes RECOMMENDED bindings for providing persistent confidentiality
2099 using MIME-based encryption schemes.

2100 12.3.6 Non-persistent Confidentiality

2101 Use of a secure network protocol such as [TLS] or [IPSEC] provides transient confidentiality of a
2102 message as it is transferred between two ebXML MSH nodes.

2103 12.3.7 Persistent Authorization

2104 The OASIS Security Services TC is actively engaged in the definition of a specification that
2105 provides for the exchange of security credentials, including NameAssertion and Entitlements that
2106 is based on [S2ML]. Use of technology that is based on this anticipated specification MAY be
2107 used to provide persistent authorization for an ebXML Message once it becomes available.
2108 ebXML has a formal liaison to this TC. There are also many ebXML member organizations and
2109 contributors that are active members of the OASIS Security Services TC such as Sun, IBM,
2110 CommerceOne, Cisco and others that are endeavoring to ensure that the specification meets the
2111 requirements of providing persistent authorization capabilities for the ebXML Message Service.

2112 12.3.8 Non-persistent Authorization

2113 Use of a secure network protocol such as [TLS] or [IPSEC] MAY be configured to provide for
2114 bilateral authentication of certificates prior to establishing a session. This provides for the ability
2115 for an ebXML MSH to authenticate the source of a connection that can be used to recognize the
2116 source as an authorized source of ebXML Messages.

2117 12.3.9 Trusted Timestamp

2118 At the time of this specification, services that offer trusted timestamp capabilities are becoming
2119 available. Once these become more widely available, and a standard has been defined for their
2120 use and expression, these standards, technologies and services will be evaluated and considered
2121 for use in providing this capability.

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
✓	Profile 0										no security services are applied to data
✓	Profile 1	✓									sending MSH applies XML/DSIG structures to message
	Profile 2		✓						✓		sending MSH authenticates and receiving MSH validates authorization from communication channel credentials
	Profile 3		✓				✓				sending MSH authenticates and receiving MSH used secure channel to transmit data
	Profile 4		✓		✓						sending MSH authenticates, the receiving MSH performs integrity checks using communications protocol
	Profile 5		✓								sending MSH authenticates the communication channel only (e.g., SSL 3.0 over TCP/IP)
	Profile 6	✓					✓				sending MSH applies XML/DSIG structures to message and passes in secure communications channel
	Profile 7	✓		✓							sending MSH applies XML/DSIG structures to message and receiving MSH returns a signed receipt
	Profile 8	✓		✓			✓				combination of profile 6 and 7
	Profile 9	✓								✓	Profile 5 with a trusted timestamp applied
	Profile 10	✓		✓						✓	Profile 9 with receiving MSH returning a signed receipt
	Profile 11	✓					✓			✓	Profile 6 with the receiving MSH applying a trusted timestamp
	Profile 12	✓		✓			✓			✓	Profile 8 with the receiving MSH applying a trusted timestamp
	Profile 13	✓				✓					sending MSH applies XML/DSIG structures to message and applies confidentiality structures (XML-Encryption)
	Profile 14	✓		✓		✓					Profile 13 with 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
	Profile 15	✓		✓						✓	sending MSH applies XML/DSIG structures to message, a trusted timestamp is added to message, receiving MSH returns a signed receipt
	Profile 16	✓				✓				✓	Profile 13 with a trusted timestamp applied
	Profile 17	✓		✓		✓				✓	Profile 14 with a trusted timestamp applied
	Profile 18	✓						✓			sending MSH applies XML/DSIG structures to message and forwards authorization credentials (S2ML)
	Profile 19	✓		✓				✓			Profile 18 with receiving MSH returning a signed receipt
	Profile 20	✓		✓				✓		✓	Profile 19 with the a trusted timestamp being applied to the sending MSH message
	Profile 21	✓		✓		✓		✓		✓	Profile 19 with the sending MSH applying confidentiality structures (XML-Encryption)
	Profile 22					✓					sending MSH encapsulates the message within confidentiality structures (XML-Encryption)

2122 **13 Synchronous and Asynchronous Responses**

2123 This section may not be needed.

2124 **14 References**

2125 <DB>What's the difference between normative and non-normative</DB>

2126 **14.1 Normative References**2127 [HTTP] RFC 2068 - Hypertext Transfer Protocol -- HTTP/1.1, R. Fielding, J. Gettys,
2128 J. Mogul, H. Frystyk, T. Berners-Lee, January 19972129 [ISO 8601] International Standards Organization Ref. ISO 8601 Second Edition,
2130 Published 19972131 [RFC 2392] IETF Request For Comments 2392. Content-ID and Message-ID Uniform
2132 Resource Locators. E. Levinson, Published August 19982133 [RFC 2396] [IETF Request For Comments 2396. Uniform Resource Identifiers \(URI\):](#)
2134 [Generic Syntax. T Berners-Lee, Published August 1998](#)

2135

2136 [RFC2045] IETF RFC 2045. Multipurpose Internet Mail Extensions (MIME) Part One:
2137 Format of Internet Message Bodies, N Freed & N Borenstein, Published
2138 November 1996

2139 [SMTP] RFC 821, Simple Mail Transfer Protocol, J Postel, August 1982

2140 [TLS] RFC2246, T. Dierks, C. Allen. January 1999.

2141 [UTF-8] UTF-8 is an encoding that conforms to ISO/IEC 10646. See [XML] for usage
2142 conventions.2143 [XML] W3C XML 1.0 Recommendation,
2144 <http://www.w3.org/TR/2000/REC-xml-20001006>2145 [XML Namespace] Recommendation for Namespaces in XML, World Wide Web Consortium, 14
2146 January 1999, <http://www.w3.org/TR/REC-xml-names>2147 **14.2 Non-Normative References**

2148 [Glossary] ebXML Glossary, see ebXML Project Team Home Page

2149 [PGP/MIME] RFC2015, "MIME Security with Pretty Good Privacy (PGP)", M. Elkins.
2150 October 1996.2151 [S/MIME] RFC2311, "S/MIME Version 2 Message Specification", S. Dusse, P.
2152 Hoffman, B. Ramsdell, L. Lundblade, L. Repka. March 1998.2153 [S/MIMECH] RFC 2312, "S/MIME Version 2 Certificate Handling", S. Dusse, P. Hoffman,
2154 B. Ramsdell, J. Weinstein. March 1998.2155 [TRPREQ] ebXML Transport, Routing and Packaging: Overview and Requirements,
2156 Version 0.96, Published 25 May 20002157 [XLINK] W3C Xlink Candidate Recommendation, <http://www.w3.org/TR/xlink/>2158 [XMLDSIG] Joint W3C/IETF XML Digital Signature specification,
2159 <http://www.w3.org/TR/2000/CR-xmlsig-core-20001031/>2160 [XMLMedia] IETF Internet Draft on XML Media Types. See [http://www.imc.org/draft-](http://www.imc.org/draft-murata-xml-08)
2161 [murata-xml-08](http://www.imc.org/draft-murata-xml-08). Note. It is anticipated that this Internet Draft will soon become
2162 a RFC. Final versions of this specification will refer to the equivalent RFC.

2163	[XMLSchema]	W3C XML Schema Candidate Recommendation,
2164		http://www.w3.org/TR/xmlschema-0/
2165		http://www.w3.org/TR/xmlschema-1/
2166		http://www.w3.org/TR/xmlschema-2/
2167	[XMTP]	XMTP - Extensible Mail Transport Protocol
2168		http://www.openhealth.org/documents/xmtp.htm

2169 **15 Disclaimer**

2170 The views and specification expressed in this document are those of the authors and are not
2171 necessarily those of their employers. The authors and their employers specifically disclaim
2172 responsibility for any problems arising from correct or incorrect implementation or use of this
2173 design.

2174 **16 Contact Information**

2175 **Team Leader**

2176 Name Rik Drummond
 2177 Company Drummond Group, Inc.
 2178 Street 5008 Bentwood Ct.
 2179 City, State, Postal Code Fort Worth, Texas 76132
 2180 Country USA
 2181 Phone +1 (817) 294-7339
 2182 EMail: rik@drummondgroup.com

2183
 2184 **Vice Team Leader**

2185 Name Chris Ferris
 2186 Company Sun Microsystems
 2187 Street One Network Drive
 2188 City, State, Postal Code Burlington, MA 01803-0903
 2189 Country USA
 2190 Phone: +1 (781) 442-3063
 2191 EMail: chris.ferris@sun.com

2192
 2193 **Team Editor**

2194 Name David Burdett
 2195 Company Commerce One
 2196 Street 4400 Rosewood Drive
 2197 City, State, Postal Code Pleasanton, CA 94588
 2198 Country USA
 2199 Phone: +1 (925) 520-4422
 2200 EMail: david.burdett@commerceone.com

2201
 2202 **Authors**

2203 Name Dick Brooks
 2204 Company Group 8760
 2205 Street 110 12th Street North, Suite F103
 2206 City, State, Postal Code Birmingham, Alabama 35203
 2207 Phone: +1 (205) 250-8053
 2208 E-mail: dick@8760.com

2209
 2210 Name David Burdett
 2211 Company Commerce One
 2212 Street 4400 Rosewood Drive
 2213 City, State, Postal Code Pleasanton, CA 94588
 2214 Country USA
 2215 Phone: +1 (925) 520-4422
 2216 EMail: david.burdett@commerceone.com

2217
 2218 Name Chris Ferris
 2219 Company Sun Microsystems
 2220 Street One Network Drive
 2221 City, State, Postal Code Burlington, MA 01803-0903
 2222 Country USA
 2223 Phone: +1 (781) 442-3063
 2224 EMail: chris.ferris@east.sun.com

2225
 2226 Name John Ibbotson
 2227 Company IBM UK Ltd

2228 Street Hursley Park
 2229 City, State, Postal Code Winchester SO21 2JN
 2230 Country United Kingdom
 2231 Phone: +44 (1962) 815188
 2232 Email: john_ibbotson@uk.ibm.com
 2233
 2234 Name Nicholas Kassem
 2235 Company Java Software, Sun Microsystems
 2236 Street 901 San Antonio Road, MS CUP02-201
 2237 City, State, Postal Code Palo Alto, CA 94303-4900
 2238 Phone: +1 (408) 863-3535
 2239 E-mail: Nick.Kassem@eng.sun.com
 2240
 2241 Name Masayoshi Shimamura
 2242 Company Fujitsu Limited
 2243 Street Shinyokohama Nikko Bldg., 15-16, Shinyokohama 2-chome
 2244 City, State, Postal Code Kohoku-ku, Yokohama 222-0033, Japan
 2245 Phone: +81-45-476-4590
 2246 E-mail: shima@rp.open.cs.fujitsu.co.jp
 2247
 2248 **Document Editing Team**
 2249 Name Ralph Berwanger
 2250 Company bTrade.com
 2251 Street 2324 Gateway Drive
 2252 City, State, Postal Code Irving, TX 75063
 2253 Country USA
 2254 Phone: +1 (972) 580-2900
 2255 EMail: rberwanger@btrade.com
 2256
 2257 Name Ian Jones
 2258 Company British Telecommunications
 2259 Street Enterprise House, 84-85 Adam Street
 2260 City, State, Postal Code Cardiff, CF24 2XF
 2261 Country United Kingdom
 2262 Phone: +44 29 2072 4063
 2263 EMail: ian.c.jones@bt.com
 2264
 2265 Name Martha Warfelt
 2266 Company Daimler Chrysler Corporation
 2267 Street 800 Chrysler Drive
 2268 City, State, Postal Code Auburn Hills, MI
 2269 Country USA
 2270 Phone: +1 (248) 944-5481 1210
 2271 EMail: maw2@daimlerchrysler.com 1211

2272 Appendix A ebXMLHeader Schema and Data Type 2273 Definitions

2274 A.1 Schema Definition

2275 The following is the definition of the **ebXMLHeader** element as a schema that conforms to
2276 [XMLSchema]. ~~DB~~The few changes from version 0.91 are highlighted.

```

2277 <?xml version = "1.0" encoding = "UTF-8"?>
2278 <xsd:schema xmlns="http://www.ebxml.org/namespaces/messageHeader"
2279 targetNamespace="http://www.ebxml.org/namespaces/messageHeader"
2280 xmlns:ds="http://www.w3.org/2000/10/xmldsig#" xmlns:xlink="http://www.w3.org/1999/xlink"
2281 xmlns:xsd="http://www.w3.org/2000/10/XMLSchema">
2282   <xsd:import namespace="http://www.w3.org/2000/10/xmldsig#"
2283     schemaLocation="http://www.w3.org/TR/2000/10/xmldsig-core-schema/xmldsig-core-schema.xsd"/>
2284
2285   <!-- EBXML HEADER -->
2286   <xsd:element name="ebXMLHeader">
2287     <xsd:complexType>
2288       <xsd:sequence>
2289         <xsd:element ref="Manifest" minOccurs="0" maxOccurs="1"/>
2290         <xsd:element ref="Header"/>
2291         <xsd:element ref="RoutingHeaderList" minOccurs="0" maxOccurs="1"/>
2292         <xsd:element ref="Acknowledgment" minOccurs="0" maxOccurs="1"/>
2293         <xsd:element ref="StatusData" minOccurs="0" maxOccurs="1"/>
2294         <xsd:element ref="ApplicationHeaders" minOccurs="0" maxOccurs="1"/>
2295         <xsd:element ref="ErrorList" minOccurs="0" maxOccurs="1"/>
2296         <xsd:element ref="ds:Signature" minOccurs="0" maxOccurs="unbounded"/>
2297       </xsd:sequence>
2298       <xsd:attribute name="version" use="fixed" value="0.930-92" type="xsd:string"/>
2299       <xsd:anyAttribute namespace="##any" processContents="lax"/>
2300     </xsd:complexType>
2301   </xsd:element>
2302
2303   <!-- MANIFEST -->
2304   <xsd:element name="Manifest">
2305     <xsd:complexType>
2306       <xsd:sequence>
2307         <xsd:element ref="Reference" maxOccurs="unbounded"/>
2308         <xsd:any namespace="##other" processContents="lax"/>
2309       </xsd:sequence>
2310       <xsd:attribute name="id" use="required" type="xsd:ID"/>
2311     </xsd:complexType>
2312   </xsd:element>
2313
2314   <xsd:element name="Reference">
2315     <xsd:complexType>
2316       <xsd:sequence>
2317         <xsd:element ref="Schema" minOccurs="0" maxOccurs="unbounded"/>
2318         <xsd:element ref="Description" minOccurs="0" maxOccurs="1"/>
2319         <xsd:any namespace="##other" processContents="lax"/>
2320       </xsd:sequence>
2321       <xsd:attribute name="id" use="required" type="xsd:ID"/>
2322     <!-- Changed required to fixed on xlink:type -->
2323     <xsd:attribute name="xlink:type" use="fixed" type="xsd:string" value="simple"/>
2324     <xsd:attribute name="xlink:href" use="required" type="xsd:uriReference"/>
2325     <!-- Changed to optional on xlink:role -->
2326     <xsd:attribute name="xlink:role" type="xsd:uriReference"/>
2327   </xsd:complexType>
2328 </xsd:element>
2329
2330   <xsd:element name="Schema">
2331     <xsd:complexType>

```

```

2333     <xsd:simpleContent>
2334         <xsd:attribute name="location" use="required" type="xsd:uriReference"/>
2335         <xsd:attribute name="version" type="xsd:string"/>
2336     </xsd:simpleContent>
2337 </xsd:complexType>
2338 </xsd:element>
2339
2340 <!-- HEADER -->
2341 <xsd:element name="Header">
2342     <xsd:complexType>
2343         <xsd:sequence>
2344             <xsd:element ref="From"/>
2345             <xsd:element ref="To"/>
2346             <xsd:element ref="CPAId"/>
2347             <xsd:element ref="ConversationId"/>
2348             <xsd:element ref="Service"/>
2349             <xsd:element ref="Action"/>
2350             <xsd:element ref="MessageData"/>
2351 <!-- Changed Reliable Messaging Inf to Quality Of Service Info. -->
2352 <!-- Removed DeliveryReceiptRequested and TimeToLive and made them optional attributes of
2353 Quality of Service Info -->
2354             <xsd:element ref="QualityOfServiceInfo" minOccurs="0" maxOccurs="1"/>
2355 <!-- Changed description from maxOccurs 1 to unbounded -->
2356             <xsd:element ref="Description" minOccurs="0" maxOccurs="unbounded"/>
2357 <!-- Added SequenceNumber element -->
2358             <xsd:element ref="SequenceNumber" minOccurs="0" maxOccurs="1"/>
2359             <xsd:any namespace="##other" processContents="lax" minOccurs="0" maxOccurs="unbounded"/>
2360         </xsd:sequence>
2361         <xsd:attribute name="id" type="xsd:ID"/>
2362     </xsd:complexType>
2363 </xsd:element>
2364
2365 <xsd:element name="To">
2366     <xsd:complexType>
2367         <xsd:simpleContent>
2368             <xsd:extension base="xsd:string">
2369                 <xsd:attribute name="type" type="xsd:string"/>
2370             </xsd:extension>
2371         </xsd:simpleContent>
2372     </xsd:complexType>
2373 </xsd:element>
2374
2375 <xsd:element name="CPAId" type="xsd:string"/>
2376
2377 <xsd:element name="ConversationId" type="xsd:string"/>
2378
2379 <xsd:element name="Service" type="xsd:string"/>
2380
2381 <xsd:element name="Action" type="xsd:string"/>
2382
2383 <xsd:element name="MessageData">
2384     <xsd:complexType>
2385         <xsd:sequence>
2386             <xsd:element ref="MessageId"/>
2387             <xsd:element ref="Timestamp"/>
2388             <xsd:element ref="RefToMessageId" minOccurs="0" maxOccurs="1"/>
2389         </xsd:sequence>
2390     </xsd:complexType>
2391 </xsd:element>
2392
2393 <xsd:element name="MessageId" type="xsd:string"/>
2394
2395 <xsd:element name="QualityOfServiceInfo">
2396     <xsd:complexType>
2397         <xsd:simpleContent>
2398             <xsd:attribute name="deliverySemantics" use="default" value="BestEffort"/>
2399             <xsd:simpleType>
2400                 <xsd:restriction base="xsd:NMTOKEN">
2401                     <xsd:enumeration value="OnceAndOnlyOnce"/>
2402                     <xsd:enumeration value="BestEffort"/>
2403                 </xsd:restriction>

```

```

2404     </xsd:simpleType>
2405 <!-- Added in messageOrderSemantics attribute -->
2406     <xsd:attribute name="messageOrderSemantics" use="default" value="NotGuaranteed"/>
2407     <xsd:simpleType>
2408         <xsd:restriction base="xsd:NMTOKEN">
2409             <xsd:enumeration value="Guaranteed"/>
2410             <xsd:enumeration value="NotGuaranteed"/>
2411         </xsd:restriction>
2412     </xsd:simpleType>
2413 <!-- Added in deliveryReceiptRequested attribute -->
2414     <xsd:attribute name="deliveryReceiptRequested" use="default" value="None"/>
2415     <xsd:simpleType>
2416         <xsd:restriction base="xsd:NMTOKEN">
2417             <xsd:enumeration value="Signed"/>
2418             <xsd:enumeration value="UnSigned"/>
2419             <xsd:enumeration value="None"/>
2420         </xsd:restriction>
2421     </xsd:simpleType>
2422 <!-- Added in timeToLive attribute -->
2423     <xsd:attribute name="timeToLive" type="xsd:timeInstant"/>
2424 </xsd:simpleContent>
2425 </xsd:complexType>
2426 </xsd:element>
2427
2428 <!-- ROUTING HEADER LIST -->
2429 <xsd:element name="RoutingHeaderList">
2430     <xsd:complexType>
2431         <xsd:sequence>
2432             <xsd:element ref="RoutingHeader" maxOccurs="unbounded"/>
2433         </xsd:sequence>
2434         <xsd:attribute name="id" type="xsd:ID"/>
2435     </xsd:complexType>
2436 </xsd:element>
2437
2438 <xsd:element name="RoutingHeader">
2439     <xsd:complexType>
2440         <xsd:sequence>
2441             <xsd:element ref="SenderURI"/>
2442             <xsd:element ref="ReceiverURI"/>
2443             <xsd:element ref="ErrorURI" minOccurs="0" maxOccurs="1"/>
2444             <xsd:element ref="Timestamp"/>
2445             <xsd:element ref="SequenceNumber" minOccurs="0" maxOccurs="1"/>
2446             <xsd:any namespace="##other" processContents="lax" minOccurs="0" maxOccurs="unbounded"/>
2447         </xsd:sequence>
2448         <xsd:attribute name="reliableMessagingMethod"/>
2449         <xsd:simpleType>
2450             <xsd:restriction base="xsd:NMTOKEN">
2451                 <xsd:enumeration value="ebXML"/>
2452                 <xsd:enumeration value="Transport"/>
2453             </xsd:restriction>
2454         </xsd:simpleType>
2455         <xsd:attribute name="intermediateAckRequested"/>
2456         <xsd:simpleType>
2457             <xsd:restriction base="xsd:NMTOKEN">
2458                 <xsd:enumeration value="Signed"/>
2459                 <xsd:enumeration value="UnSigned"/>
2460                 <xsd:enumeration value="None"/>
2461             </xsd:restriction>
2462         </xsd:simpleType>
2463     </xsd:complexType>
2464 </xsd:element>
2465
2466 <xsd:element name="SenderURI" type="xsd:uriReference"/>
2467
2468 <xsd:element name="ReceiverURI" type="xsd:uriReference"/>
2469
2470 <xsd:element name="SequenceNumber" type="xsd:positiveInteger" minOccurs="0" maxOccurs="1"/>
2471
2472 <xsd:element name="ErrorURI" type="xsd:uriReference" minOccurs="0" maxOccurs="1"/>
2473

```

```

2474 <!-- APPLICATION HEADERS -->
2475 <xsd:element name="ApplicationHeaders" type="ApplicationHeaders"/>
2476 <xsd:complexType name="ApplicationHeaders">
2477   <xsd:sequence>
2478     <xsd:any namespace="##other" processContents="lax"/>
2479   </xsd:sequence>
2480   <xsd:attribute name="id" type="xsd:ID"/>
2481 </xsd:complexType>
2482
2483 <!-- ACKNOWLEDGEMENT -->
2484 <xsd:element name="Acknowledgment">
2485   <xsd:complexType>
2486     <xsd:sequence>
2487       <xsd:element ref="Timestamp"/>
2488       <xsd:element ref="From" minOccurs="0" maxOccurs="1"/>
2489     </xsd:sequence>
2490     <xsd:attribute name="id" type="xsd:ID"/>
2491     <xsd:attribute name="type" use="default" value="DeliveryReceipt"/>
2492     <xsd:simpleType>
2493       <xsd:restriction base="xsd:NMTOKEN">
2494         <xsd:enumeration value="DeliveryReceipt"/>
2495         <xsd:enumeration value="IntermediateAck"/>
2496       </xsd:restriction>
2497     </xsd:simpleType>
2498     <xsd:attribute name="signed" type="xsd:boolean"/>
2499   </xsd:complexType>
2500 </xsd:element>
2501
2502 <!-- ERROR LIST -->
2503 <xsd:element name="ErrorList">
2504   <xsd:complexType>
2505     <xsd:sequence>
2506       <xsd:element ref="Error" maxOccurs="unbounded"/>
2507     </xsd:sequence>
2508     <xsd:attribute name="id" type="xsd:ID"/>
2509     <xsd:attribute name="highestSeverity" use="default" value="Warning"/>
2510     <xsd:simpleType>
2511       <xsd:restriction base="xsd:string">
2512         <xsd:enumeration value="Warning"/>
2513         <xsd:enumeration value="Error"/>
2514       </xsd:restriction>
2515     </xsd:simpleType>
2516   </xsd:complexType>
2517 </xsd:element>
2518
2519 <xsd:element name="Error">
2520   <xsd:complexType>
2521     <xsd:attribute name="codeContext" use="required" type="xsd:uriReference"/>
2522     <xsd:attribute name="errorCode" use="required" type="xsd:string"/>
2523     <xsd:attribute name="severity" use="default" value="Warning"/>
2524     <xsd:simpleType>
2525       <xsd:restriction base="xsd:NMTOKEN">
2526         <xsd:enumeration value="Warning"/>
2527         <xsd:enumeration value="Error"/>
2528       </xsd:restriction>
2529     </xsd:simpleType>
2530     <xsd:attribute name="location" type="xsd:string"/>
2531     <xsd:attribute name="xml:lang" type="xsd:language"/>
2532     <xsd:attribute name="errorMessage" type="xsd:string"/>
2533     <xsd:attribute name="softwareDetails" type="xsd:string"/>
2534   </xsd:complexType>
2535 </xsd:element>
2536
2537 <!-- STATUS DATA -->
2538 <xsd:element name="StatusData">
2539   <xsd:sequence>
2540     <xsd:element ref="RefToMessageId"/>
2541     <xsd:element ref="Timestamp" minOccurs="0" maxOccurs="1"/>
2542     <xsd:element name="ForwardURI" type="xsd:uriReference" minOccurs="0" maxOccurs="1"/>
2543   </xsd:sequence>
2544   <xsd:attribute name="messageStatus"/>

```

```
2545     <xsd:simpleType>
2546         <xsd:restriction base="xsd:NMTOKEN">
2547             <xsd:enumeration value="Unauthorized"/>
2548             <xsd:enumeration value="NotRecognized"/>
2549             <xsd:enumeration value="Received"/>
2550             <xsd:enumeration value="Processed"/>
2551             <xsd:enumeration value="Forwarded"/>
2552         </xsd:restriction>
2553     </xsd:simpleType>
2554 </xsd:element>
2555
2556 <!-- COMMON ELEMENTS -->
2557 <xsd:element name="From">
2558     <xsd:complexType>
2559         <xsd:simpleContent>
2560             <xsd:extension base="xsd:string">
2561                 <xsd:attribute name="type" type="xsd:string"/>
2562             </xsd:extension>
2563         </xsd:simpleContent>
2564     </xsd:complexType>
2565 </xsd:element>
2566
2567 <xsd:element name="Description">
2568     <xsd:complexType>
2569         <xsd:simpleContent>
2570             <xsd:extension base="xsd:string">
2571                 <xsd:attribute name="xml:lang" type="xsd:NMTOKEN"/>
2572             </xsd:extension>
2573         </xsd:simpleContent>
2574     </xsd:complexType>
2575 </xsd:element>
2576
2577 <xsd:element name="RefToMessageId" type="xsd:string"/>
2578
2579 <xsd:element name="Timestamp" type="xsd:timeInstant"/>
2580 <!-- Does timeInstant conform to ISO 2601? -->
2581
2582 </xsd:schema>
```

2583 A.2 Data Type Definition

2584 This section will contain a [XML] DTD that is equivalent to the schema defined in section A.1.

2585 **Appendix B Examples**

2586 To be completed.

2587 **Appendix C Communication Protocol Interfaces**

2588 This Appendix describes how the ebXML Message Service messages are carried by
 2589 Communication Protocols. Two protocols are supported:

- 2590 • Hypertext Transfer Protocol – HTTP/1.1, in both asynchronous and synchronous forms,
 2591 and
- 2592 • SMTP – Simple Mail Transfer Protocol

2593 **C.1 HTTP**

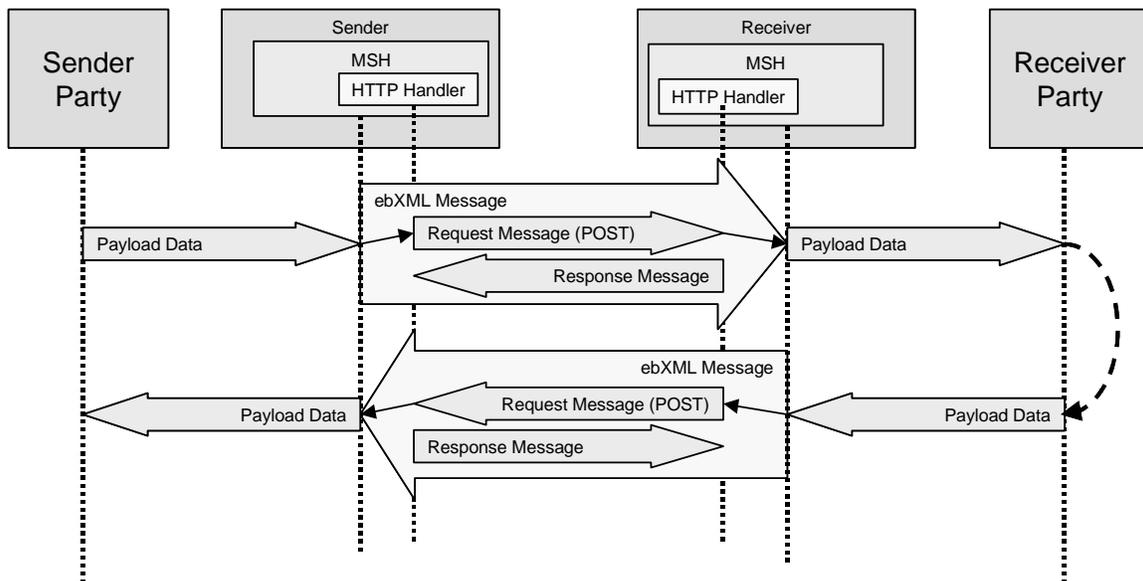
2594 This section describes how to transport ebXML compliant messages of [HTTP]. This can work in
 2595 one of the following two ways:

- 2596 • asynchronously, where the response to a message is sent using a separate HTTP POST,
 2597 and
- 2598 • synchronously, where the response to a message is sent on the HTTP RESPONSE
 2599 returned from an HTTP POST

2600 These are described below.

2601 **C.1.1 Asynchronous HTTP**

2602 In Asynchronous HTTP, all ebXML Message Service messages are carried by an HTTP Request
 2603 Message (POST method). The HTTP Response Message to an HTTP Request Message has no
 2604 entity body. This is illustrated by the figure below.



2605
 2606 **Figure C.1 Asynchronous HTTP Message Flow**

2607 A message that is being sent asynchronously MAY be identified by the following HTTP header:

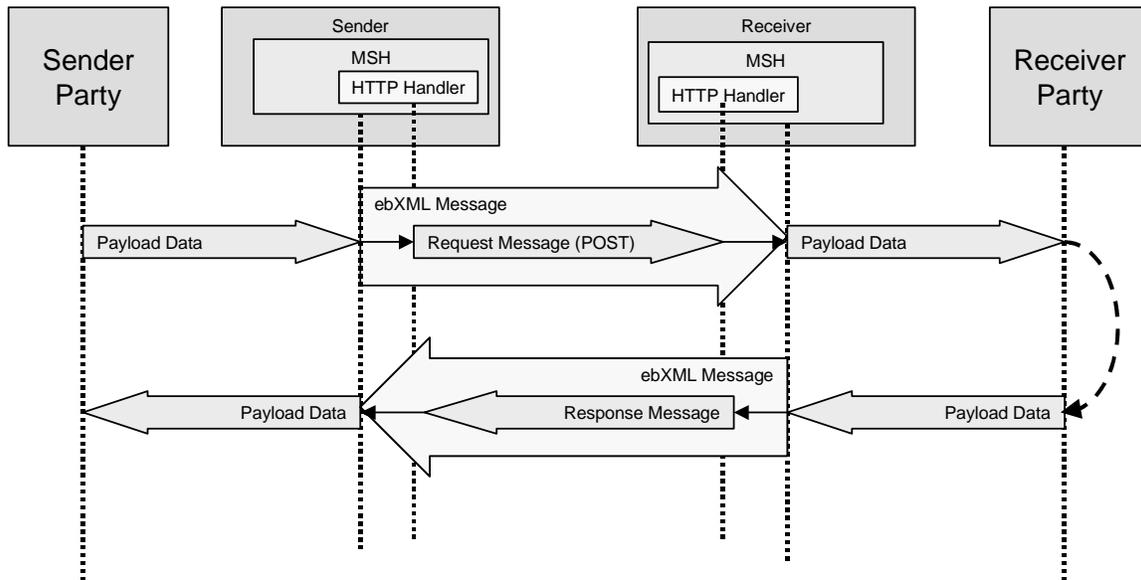
2608 `ebxmlresponse=asynchronous`

2609 If the `ebXMLresponse` HTTP parameter is omitted then it MUST be assumed that the response
 2610 is sent asynchronously.

2611 **C.1.2 Synchronous HTTP**

2612 *[The Synchronous HTTP section has not been agreed to by the membership of the TRP*
 2613 *Project Team; however, it is being included to provide a basis for POC developers of MSH*
 2614 *implementations. Implementers MUST be prepared for some change to the content of this*
 2615 *section.]*

2616 In Synchronous HTTP, one ebXML Message Service message is carried by an HTTP Request
 2617 Message (POST method) with the ebXML Message that is a response to the first message sent
 2618 in the HTTP Response Message to the HTTP Request Message. This is illustrated by the figure
 2619 below.



2620
 2621 **Figure C.2 Synchronous HTTP Message Flow**

2622 If a response is being sent synchronously, the following HTTP header MUST be included in the
 2623 HTTP envelope:

2624 `ebxmlresponse=synchronous`

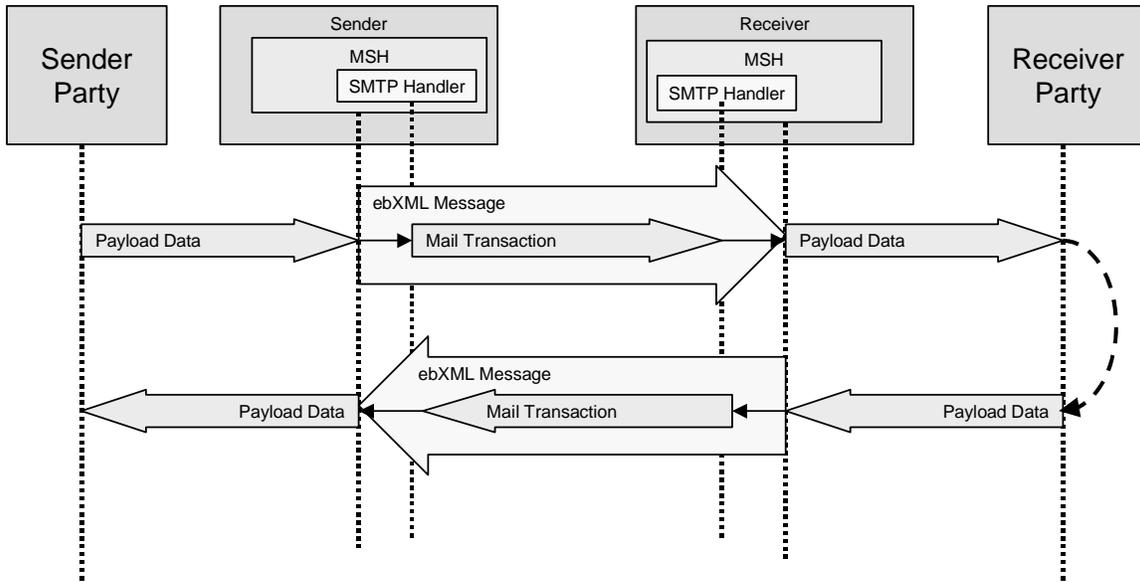
2625 **C.1.3 Use of Error Codes**

2626 Communication Protocol Error Codes are used only to report errors in the communication
 2627 protocol envelope (see section 7.1). A normal OK Response (e.g. an HTTP code 200) is used
 2628 even if there are errors in the MIME envelope, the ebXML Header document or the payload.

2629

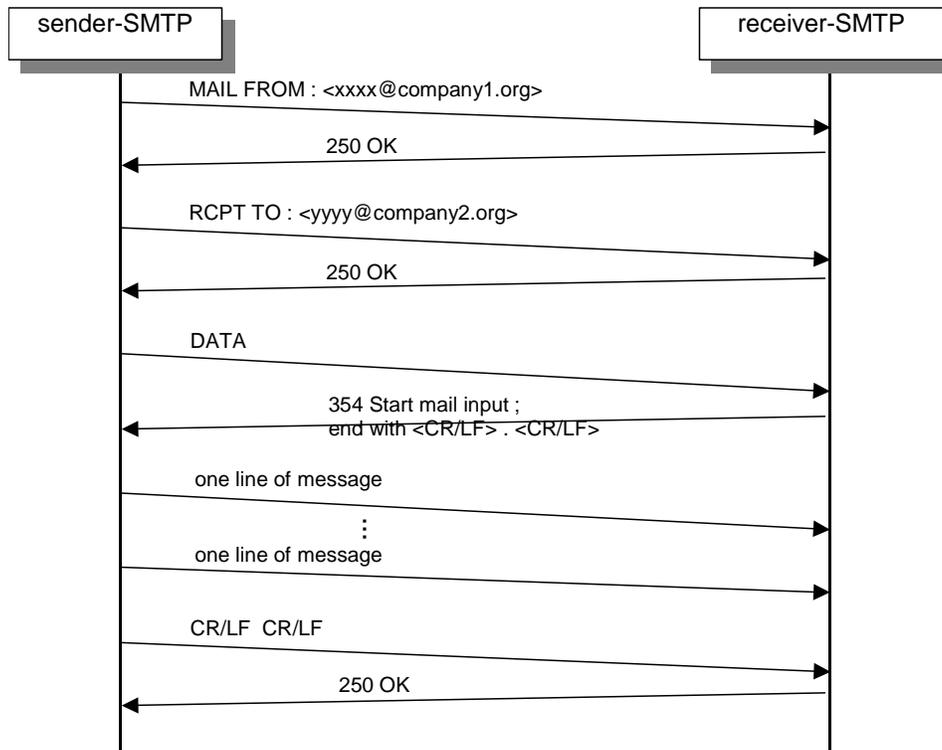
2630 **C.2 SMTP**

2631 All ebXML Message Service messages are carried as mail in an [SMTP] Mail Transaction as
 2632 shown in the figure below.



2633
 2634 **Figure C.3 SMTP Message Flow**

2635 The Mail Transaction follows RFC 821, "SIMPLE MAIL TRANSFER PROTOCOL", as shown in
 2636 the following Figure:



2637
 2638 **Figure C.4 SMTP Sequence**

2639 **~~C.3FTP~~**

2640 ~~This section will describe how ebXML Messages may be sent using the File~~
2641 ~~Transfer protocol as defined in RFC 959~~

2642 ~~This section is to be completed.~~

2643 **~~C.4Communication Protocol Errors~~**2644 **~~C.4.1Use of Error Codes~~**

2645 ~~Communication Protocol Error Codes are used only to report errors in the~~
2646 ~~communication protocol envelope (see section 7.1). A normal OK~~
2647 ~~Response (e.g. an HTTP code 200) is used even if there are errors in~~
2648 ~~the MIME envelope, the ebXML Header document or the payload.~~

2649 **C.4.2C.3 Communication Errors during Reliable Messaging**

2650 When the Sender or the Receiver detects a transport protocol level error (such as an HTTP,
2651 SMTP or FTP error) and Reliable Messaging is being used then the appropriate transport
2652 recovery handler will execute a recovery sequence. Only if the error is unrecoverable, does
2653 Reliable Messaging recovery take place (see section 10).

2654 **Appendix D ~~Reliable Messaging Processing~~**
2655 **LogicRequest for MIME media type**
2656 **Application/Vendor Tree - vnd**

2657 This section ~~will contain non-normative reference processing logic to describe the behavior of a~~
2658 ~~MSH that is taking part in reliable messaging. It's purpose is to assist implementers in developing~~
2659 ~~consistent interoperable solutions.~~is non-normative. It contains the information forwarded to
2660 IANA to register the MIME subtype vnd.be+xml. The information was extracted verbatim from the
2661 e-mail message forward by Dick Brooks, Group8760, on behalf of the ebXML Transportation,
2662 Routing, and Packaging Project Team.

2663
2664 From: Dick Brooks [mailto:dick@8760.com]
2665 Sent: Thursday, February 01, 2001 2:00 PM
2666 To: iana@iana.org; Dick Brooks
2667 Subject: Request for MIME media type Application/Vendor Tree - vnd.
2668

2669 Name: Richard Brooks (on behalf of OASIS and UN/CEFACT)
2670 E-mail: dick@8760.com
2671 MIME media type name: Application
2672 MIME subtype name: Vendor Tree - vnd.eb+xml
2673 Required parameters: version
2674 Optional parameters: charset
2675 Encoding considerations: N/A
2676
2677 Security considerations: N/A
2678 Interoperability considerations: N/A
2679 Published specification: Message Service Specification ebXML Transport,
2680 Routing and Packaging
2681 Applications that use this media: ebXML Message Handling Services
2682 Additional information:
2683 1. Magic number(s): N/A
2684 2. File extension(s): .ebx
2685 3. Macintosh file type code: N/A
2686 4. Object Identifiers: N/A
2687

2688 This media type is owned jointly by OASIS, UN/CEFACT and ebXML
2689 Person to contact for further information:
2690 1. Name: Richard Brooks
2691 2. E-mail: dick@8760.com
2692 Intended usage: Common
2693 Identifies ebXML header documents
2694 Author/Change controller:
2695 Christopher Ferris chris.ferris@east.sun.com
2696 Rik Drummond rvd2@worldnet.att.net
2697

2697

2698

|

2699 **Copyright Statement**

2700 This document and translations of it may be copied and furnished to others, and derivative works
2701 that comment on or otherwise explain it or assist in its implementation may be prepared, copied,
2702 published and distributed, in whole or in part, without restriction of any kind, provided that the
2703 above copyright notice and this paragraph are included on all such copies and derivative works.
2704 However, this document itself may not be modified in any way, such as by removing the copyright
2705 notice or references to the Internet Society or other Internet organizations, except as needed for
2706 the purpose of developing Internet standards in which case the procedures for copyrights defined
2707 in the Internet Standards process must be followed, or as required to translate it into languages
2708 other than English.

2709 The limited permissions granted above are perpetual and will not be revoked by ebXML or its
2710 successors or assigns.

2711 This document and the information contained herein is provided on an "AS IS" basis and ebXML
2712 DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO
2713 ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE
2714 ANY RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A
2715 PARTICULAR PURPOSE.