



Creating A Single Global Electronic Market

Message Service Specification

ebXML Transport, Routing & Packaging

Version 0.98b

13 March 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.

Note: implementers of this specification should consult the ebXML web site for current status and revisions to the specification (<http://www.ebxml.org>).

This version

http://www.ebxml.org/project_teams/transport/ebxml_message_service_specification_v-0.98b.pdf

Latest version

http://www.ebxml.org/project_teams/transport/ebxml_message_service_specification_v-0.98.pdf

Previous version

http://www.ebxml.org/project_teams/transport/ebxml_message_service_specification_v-0.8_001110.pdf

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.
Andrew Eisenberg – Progress Software
Colleen Evans—Progress / Sonic Software
David Fischer, Drummond Group, Inc.
Christopher Ferris – Sun Microsystems
Robert Fox - Softshare
Maryann Hondo – IBM
Jim Hughes – Fujitsu
John Ibbotson – IBM
Ian Jones – British Telecommunications
Ravi Kacker – Kraft Foods
Henry Lowe – OMG
Jim McCarthy – webXI
Bob Miller – GXS
Dale Moberg – Sterling Commerce
Joel Munter – Intel
Shumpei Nakagaki – NEC Corporation
Farrukh Najmi – Sun Microsystems
Akira Ochi – Fujitsu
Martin Sachs, IBM
Saikat Saha – Commerce One, Inc.
Masayoshi Shimamura – Fujitsu
Prakash Sinha – Netfish Technologies
Rich Salz – Zolera Systems
Tae Joon Song – eSum Technologies, Inc.
Kathy Spector – Extricity
Nikola Stojanovic – Encoda Systems, Inc.
David Turner - Microsoft
Gordon Van Huizen – Progress Software
Martha Warfelt – DaimlerChrysler
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	8
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	SOAP Structural Conformance	14
7.2	Message Package	15
7.3	Header Container	15
7.3.1	Content-Type	15
7.3.2	Header Container Example	15
7.4	Payload Container	16
7.4.1	Example of a Payload Container	16
7.5	Additional MIME Parameters	16
7.6	Reporting MIME Errors	16
8	ebXML SOAP Extensions	17
8.1	XML Prolog	17
8.1.1	XML Declaration	17
8.1.2	Encoding Declaration	17
8.2	ebXML SOAP Envelope Extensions	17
8.2.1	Namespace pseudo attribute	18
8.2.2	ebXML SOAP Extensions	18
8.2.3	#wildcard element content	18
8.3	SOAP Header element	18
8.4	MessageHeader element	19
8.4.1	From and To elements	19
8.4.2	CPAId element	19
8.4.3	ConversationId element	20
8.4.4	Service element	20
8.4.5	Action element	21
8.4.6	MessageData element	21
8.4.7	QualityOfServiceInfo element	22
8.4.8	SequenceNumber element	23
8.4.9	Description element	24

8.4.10	version attribute	24
8.4.11	SOAP mustUnderstand attribute	24
8.4.12	MessageHeader sample.....	24
8.5	TraceHeaderList element.....	24
8.5.1	SOAP mustUnderstand attribute	25
8.5.2	version attribute	25
8.5.3	TraceHeader element.....	25
8.5.4	Single Hop TraceHeader Sample	26
8.5.5	Multi-hop TraceHeader Sample.....	26
8.6	Via element	27
8.6.1	SOAP mustUnderstand attribute	28
8.6.2	SOAP actor attribute.....	28
8.6.3	version attribute	28
8.6.4	syncReply attribute	28
8.6.5	reliableMessagingMethod attribute.....	28
8.6.6	ackRequested attribute.....	28
8.6.7	CPAId element	29
8.6.8	Service and Action elements	29
8.6.9	Sample Via element	29
8.7	ErrorList element.....	29
8.7.1	id attribute.....	29
8.7.2	SOAP mustUnderstand attribute	29
8.7.3	version attribute	30
8.7.4	highestSeverity attribute	30
8.7.5	Error element.....	30
8.7.6	Examples.....	31
8.7.7	errorCode values	31
8.7.8	Reporting Errors in the ebXML Elements	31
8.7.9	Non-XML Document Errors	32
8.8	Signature element.....	32
8.9	SOAP Body Extensions	32
8.10	Manifest element	33
8.10.1	id attribute.....	33
8.10.2	SOAP mustUnderstand attribute	33
8.10.3	version attribute	33
8.10.4	#wildcard element.....	33
8.10.5	Reference element	33
8.10.6	What References are Included in a Manifest	34
8.10.7	Manifest Validation	35
8.10.8	Manifest sample	35
8.11	StatusData Element.....	35
8.11.1	RefToMessageId element	35
8.11.2	Timestamp element	35
8.11.3	SOAP mustUnderstand attribute	35
8.11.4	version attribute	35
8.11.5	messageStatus attribute	36
8.11.6	StatusData sample	36
8.12	Acknowledgment Element	36
8.12.1	Timestamp element	36
8.12.2	From element	36
8.12.3	SOAP mustUnderstand attribute	36
8.12.4	version attribute	37
8.12.5	type attribute.....	37
8.12.6	signed attribute	37

- 8.12.7 Acknowledgement sample..... 37
- 8.13 Combining ebXML SOAP Extension Elements 37
 - 8.13.1 Manifest element 37
 - 8.13.2 MessageHeader element 37
 - 8.13.3 TraceHeaderList element 37
 - 8.13.4 StatusData element 38
 - 8.13.5 ErrorList element 38
 - 8.13.6 Acknowledgment element 38
 - 8.13.7 Signature element 38
 - 8.13.8 Via element..... 38
- 9 Message Service Handler Services 39
 - 9.1 Message Status Request Service..... 39
 - 9.1.1 Message Status Request Message 39
 - 9.1.2 Message Status Response Message 39
 - 9.1.3 Security Considerations..... 40
 - 9.2 Message Service Handler Ping Service 40
 - 9.2.1 Message Service Handler Ping Message..... 40
 - 9.2.2 Message Service Handler Pong Message..... 41
 - 9.2.3 Security Considerations..... 41
- 10 Reliable Messaging..... 42
 - 10.1.1 Persistent Storage and System Failure 42
 - 10.1.2 Methods of Implementing Reliable Messaging 42
 - 10.2 Reliable Messaging Parameters..... 42
 - 10.2.1 Delivery Semantics..... 42
 - 10.2.2 MSHTimeAccuracy..... 43
 - 10.2.3 Time To Live..... 43
 - 10.2.4 reliableMessagingMethod..... 43
 - 10.2.5 AckRequested 43
 - 10.2.6 Timeout Parameter..... 44
 - 10.2.7 Retries 44
 - 10.2.8 RetryInterval 44
 - 10.2.9 PersistDuration..... 44
 - 10.3 ebXML Reliable Messaging Protocol 44
 - 10.4 Failed Message Delivery 48
- 11 Error Reporting and Handling 49
 - 11.1 Definitions 49
 - 11.2 Types of Errors 49
 - 11.3 When to generate Error Messages 49
 - 11.3.1 Security Considerations..... 50
 - 11.4 Identifying the Error Reporting Location..... 50
 - 11.5 Service and Action Element Values 50
- 12 Security 51
 - 12.1 Security and Management..... 51
 - 12.2 Collaboration Protocol Agreement 51
 - 12.3 Countermeasure Technologies 51
 - 12.3.1 Persistent Digital Signature 51
 - 12.3.2 Persistent Signed Receipt 53
 - 12.3.3 Non-persistent Authentication..... 53
 - 12.3.4 Non-persistent Integrity 54
 - 12.3.5 Persistent Confidentiality..... 54

12.3.6	Non-persistent Confidentiality.....	54
12.3.7	Persistent Authorization.....	54
12.3.8	Non-persistent Authorization	54
12.3.9	Trusted Timestamp.....	54
13	References.....	57
13.1	Normative References.....	57
13.2	Non-Normative References	58
14	Disclaimer	59
15	Contact Information.....	60
Appendix A ebXML SOAP Extension Elements Schema		62
Appendix B Communication Protocol Bindings.....		67
B.1	Introduction	67
B.2	HTTP.....	67
B.2.1	Minimum level of HTTP protocol	67
B.2.2	Sending ebXML Service messages over HTTP	67
B.2.3	HTTP Response Codes.....	69
B.2.4	SOAP Error conditions and Synchronous Exchanges	69
B.2.5	Synchronous vs. Asynchronous	69
B.2.6	Access Control.....	69
B.2.7	Confidentiality and Communication Protocol Level Security.....	70
B.3	SMTP	70
B.3.1	Minimum level of supported protocols.....	71
B.3.2	Sending ebXML Messages over SMTP	71
B.3.3	Response Messages	72
B.3.4	Access Control.....	73
B.3.5	Confidentiality and Communication Protocol Level Security.....	73
B.3.6	SMTP Model.....	73
B.4	Communication Errors during Reliable Messaging	74
Copyright Statement		75

1 4 Introduction

2 This is a draft standard for trial implementation. This specification is 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 SHALL
7 either be incorporated into a future version of this specification or referenced as an external
8 specification as deemed most suitable by the ebXML Transport, Routing and Packaging Team.

9 4.1 Summary of Contents of Document

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

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

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

19 This specification is organized around the following topics:

- 20 • **Packaging Specification** – A description of how to package an ebXML Message and its
21 associated parts into a form that can sent using a communications protocol such as
22 HTTP or SMTP (section 7)
- 23 • **ebXML SOAP Extensions** – A specification of the structure and composition of the
24 information necessary for an *ebXML Message Service* to successfully generate or
25 process an ebXML Message (section 8)
- 26 • **Message Service Handler Services** – A description of two services that enable one
27 service to discover the status of another Message Service Handler (MSH) or an individual
28 message (section 9)
- 29 • **Reliable Messaging** – The Reliable Messaging function defines an interoperable
30 protocol such that any two Message Service implementations can “reliably” exchange
31 messages that are sent using “reliable messaging” once-and-only-once delivery
32 semantics (section 10)
- 33 • **Error Handling** – This section describes how one *ebXML Message Service* reports
34 errors it detects to another ebXML Message Service Handler (section 11)
- 35 • **Security** – This provides a specification of the security semantics for ebXML Messages
36 (section12).

37 Appendices to this specification cover the following:

- 38 • **Appendix A Schema** – This normative appendix contains [XML Schema] for the ebXML
39 Header document.
- 40 • **Appendix B Communication Protocol Envelope Mappings** – This normative appendix
41 describes how to transport *ebXML Message Service* compliant messages over [HTTP]
42 and [SMTP]

43 4.2 Document Conventions

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

46 MIME components. Notes are listed in Times New Roman font and are informative (non-
47 normative).

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

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

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

73 4.3 Audience

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

76 4.4 Caveats and Assumptions

77 It is assumed that the reader has an understanding of transport protocols, MIME, XML, SOAP,
78 SOAP Messages with Attachments and security technologies.

79 4.5 Related Documents

80 The following set of related specifications are developed independent of this specification as part
81 of the ebXML initiative:

- 82 • **ebXML Message Services Requirements Specification [EBXMLMSREQ]** – defines the
83 requirements of these Message Services
- 84 • **ebXML Technical Architecture [EBXMLTA]** – defines the overall technical architecture
85 for ebXML
- 86 • **ebXML Technical Architecture Security Specification [EBXMLTASEC]** – defines the
87 security mechanisms necessary to negate anticipated, selected threats
- 88 • **ebXML Collaboration Protocol Profile and Agreement Specification [EBXMLTP]**
89 (under development) - defines how one party can discover and/or agree upon the
90 information that party needs to know about another party prior to sending them a
91 message that complies with this specification

92
93

- **ebXML Registry/Repository Services Specification [EBXMLRSS]** – defines a registry service for the ebXML environment

94 **5 Design Objectives**

95 The design objectives of this specification are to define a wire format and protocol for a Message
96 Service to support XML-based electronic business between small, medium, and large
97 enterprises. While the specification has been primarily designed to support XML-based electronic
98 business, the authors of the specification have made every effort to ensure that the exchange of
99 non-XML business information is fully supported. This specification is intended to enable a low
100 cost solution, while preserving a vendor's ability to add unique value through added robustness
101 and superior performance. It is the intention of the Transport, Routing and Packaging Project
102 Team to keep this specification as straightforward and succinct as possible.

103 Every effort has been made to ensure that the REQUIRED functionality described in this
104 specification has been prototyped by the ebXML Proof of Concept Team in order to ensure the
105 clarity, accuracy and efficiency of this specification.

106 6 System Overview

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

112 The *ebXML Message Service* is defined as a set of layered extensions to the base Simple Object
113 Access Protocol [SOAP] and SOAP Messages with Attachments [SOAPATTACH] specifications
114 that have a broad industry acceptance, and that serve as the foundation of the work of the W3C
115 XML Protocol Core working group. The *ebXML Message Service* provides the security and
116 reliability features necessary to support international electronic business that are not provided in
117 the SOAP and SOAP Messages with Attachments specifications.

118 6.1 What the Message Service does

119 The *ebXML Message Service* defines robust, yet basic, functionality to transfer messages
120 between trading parties using various existing communication protocols. The *ebXML Message*
121 *Service* is structured to allow for messaging reliability, persistence, security and extensibility.

122 The *ebXML Message Service* is provided for environments requiring a robust, yet low cost
123 solution to enable electronic business. It is one of the four "infrastructure" components of ebXML.
124 The other three are: Registry/Repository [EBXMLRSS], Collaboration Protocol Profile/Agreement
125 [EBXMLTP] and ebXML Technical Architecture [EBXMLTA].

126 6.2 Message Service Overview

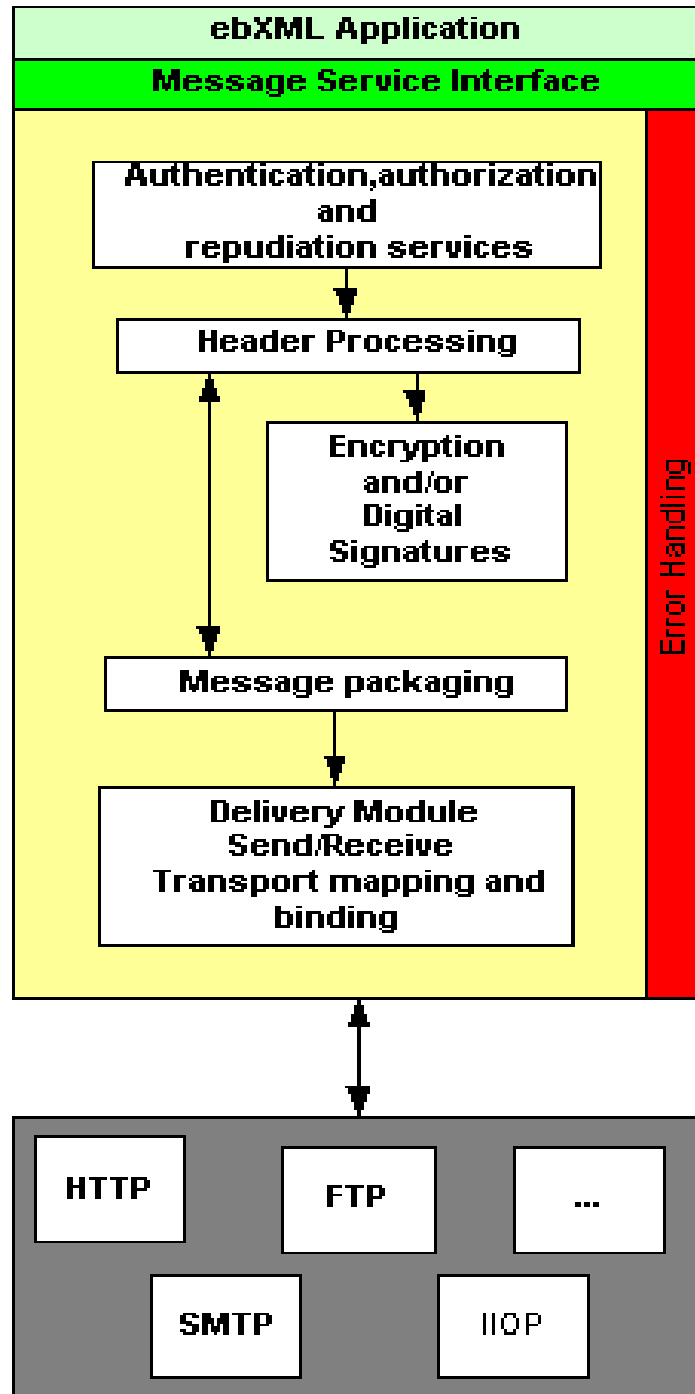
127 The *ebXML Message Service* may be conceptually broken down into following three parts: (1) an
128 abstract *Service Interface*, (2) functions provided by the Message Service Handler (MSH), and (3)
129 the mapping to underlying transport service(s).

130 The following diagram depicts a logical arrangement of the functional modules that exist within
131 one possible implementation of the *ebXML Message Services* architecture. These modules are
132 arranged in a manner to indicate their inter-relationships and dependencies.

- 133 • **Header Processing** - the creation of the ebXML Header elements for the *ebXML*
134 *Message* uses input from the application, passed through the Message Service Interface,
135 information from the *Collaboration Protocol Agreement* (CPA defined in [EBXMLTP]) that
136 governs the message, and generated information such as digital signature, timestamps
137 and unique identifiers.
- 138 • **Header Parsing** - extracting or transforming information from a received ebXML Header
139 element into a form that is suitable for processing by the MSH implementation.
- 140 • **Security Services** - digital signature creation and verification, authentication and
141 authorization. These services MAY be used by other components of the MSH including
142 the Header Processing and Header Parsing components.
- 143 • **Reliable Messaging Services** - handles the delivery and acknowledgment of ebXML
144 Messages sent with *deliverySemantics* of **OnceAndOnlyOnce**. The service includes
145 handling for persistence, retry, error notification and acknowledgment of messages
146 requiring reliable delivery.
- 147 • **Message Packaging** - the final enveloping of an *ebXML Message* (ebXML header
148 elements and payload) into its SOAP Messages with Attachments [SOAPATTACH]
149 container.
- 150 • **Error Handling** - this component handles the reporting of errors encountered during
151 MSH or Application processing of a message.

152
153
154
155

- **Message Service Interface** - an abstract service interface that applications use to interact with the MSH to send and receive messages and which the MSH uses to interface with applications that handle received messages.



156
157

Figure 6-1 Typical Relationship between ebXML Message Service Handler Components

158 7 Packaging Specification

159 7.1 Introduction

160 An ebXML Message is a communication protocol independent MIME/Multipart message
 161 envelope, structured in compliance with the SOAP Messages with Attachments [SOAPATTACH]
 162 specification, referred to as a *Message Package*.

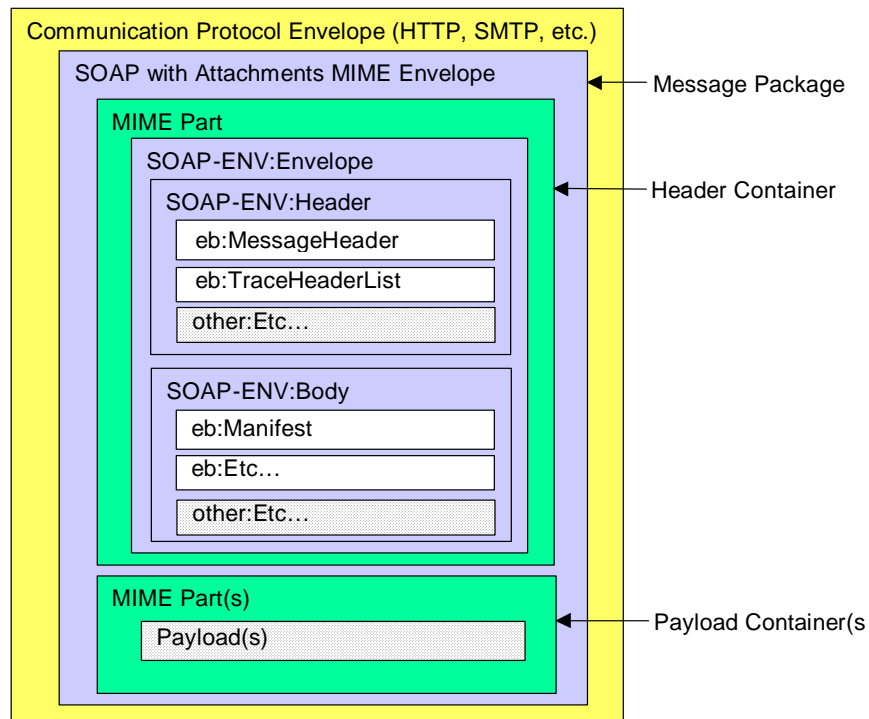
163 There are two logical MIME parts within the *Message Package*:

- 164 • A MIME part, referred to as the *Header Container*, containing one SOAP 1.1 compliant
 165 message. This XML document is referred to as a *SOAP Message* for the remainder of
 166 this specification,
- 167 • zero or more MIME parts, referred to as *Payload Containers*, containing application level
 168 payloads.

169 The *SOAP Message* is an XML document that consists of the SOAP Envelope element. This is
 170 the root element of the XML document representing the *SOAP Message*. The SOAP Envelope
 171 element consists of the following:

- 172 • One SOAP Header element. This is a generic mechanism for adding features to a *SOAP*
 173 *Message*, including ebXML specific header elements.
- 174 • One SOAP Body element. This is a container for message service handler control data
 175 and information related to the payload parts of the message.

176 The general structure and composition of an ebXML Message is described in the following figure.



177 **Figure 7-1 ebXML Message Structure**

178 7.1.1 SOAP Structural Conformance

179 *ebXML Message* packaging SHALL comply with the following specifications:

- 180 • Simple Object Access Protocol (SOAP) 1.1 [SOAP]
181 • SOAP Messages with Attachments [SOAPATTACH]
- 182 Carrying ebXML headers in *SOAP Messages* does not mean that ebXML overrides existing
183 semantics of SOAP, but rather that the semantics of ebXML over SOAP maps directly onto SOAP
184 semantics.

185 7.2 Message Package

186 All MIME header elements of the *Message Package* MUST be in conformance with the SOAP
187 Messages with Attachments [SOAPATTACH] specification. In addition, the `Content-Type`
188 MIME header in the *Message Package* MUST contain a `type` attribute that matches the MIME
189 media type of the MIME body part that contains the *SOAP Message* document. In accordance
190 with the [SOAP] specification, the MIME media type of the *SOAP Message* MUST have the value
191 "text/xml."

192 It is strongly RECOMMENDED that the root part contain a `Content-ID` MIME header structured
193 in accordance with [RFC2045], and that in addition to the required parameters for the
194 Multipart/Related media type, the `start` parameter (OPTIONAL in [RFC2387]) always be
195 present. This permits more robust error detection. For example:

```
196  
197 Content-Type: multipart/related; type="text/xml"; boundary="-----boundaryValue";  
198 start="<cid-of-SOAP-message-body-part>"
```

199 7.3 Header Container

200 The root body part of the *Message Package* is referred to in this specification as the *Header*
201 *Container*. The *Header Container* is a MIME body part that MUST consist of one *SOAP Message*
202 as defined in the SOAP Messages with Attachments [SOAPATTACH] specification.

203 7.3.1 Content-Type

204 The MIME `Content-Type` header for the *Header Container* MUST have the value
205 "text/xml" in accordance with the [SOAP] specification. The `Content-Type` header MAY
206 contain a "charset" attribute. For example:

```
207  
208 Content-Type: text/xml; charset="UTF-8"
```

209 7.3.1.1 charset Attribute

210 The MIME `charset` attribute identifies the character set used to create the *SOAP Message*.
211 The semantics of this attribute are described in the "charset parameter / encoding considerations"
212 of `text/xml` as specified in [XMLMedia]. The list of valid values can be found at
213 <http://www.iana.org/>.

214 If both are present, the MIME `charset` attribute SHALL be equivalent to the encoding
215 declaration of the *SOAP Message*. If provided, the MIME `charset` attribute MUST NOT contain
216 a value conflicting with the encoding used when creating the *SOAP Message*.

217 For maximum interoperability it is RECOMMENDED that [UTF-8] be used when encoding this
218 document. Due to the processing rules defined for media types derived from `text/xml`
219 [XMLMedia], this MIME attribute has no default. For example:

```
220  
221 charset="UTF-8";
```

222 7.3.2 Header Container Example

223 The following fragment represents an example of a *Header Container*.

```
224  
225 Content-ID: messagepackage-123@example.com     -- | Header Container
```

```

226 Content-Type: text/xml;
227         charset="UTF-8"
228
229 <SOAP-ENV:Envelope |--SOAP Message
230     xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
231     <SOAP-ENV:Header>
232     ...
233     </SOAP-ENV:Header>
234     <SOAP-ENV:Body>
235     ...
236     </SOAP-ENV:Body>
237 </SOAP-ENV:Envelope>
    
```

238 **7.4 Payload Container**

239 Zero or more *Payload Containers* MAY be present within a *Message Package* in conformance
 240 with the SOAP Messages with Attachments [SOAPATTACH] specification.

241 If the *Message Package* contains an application payload, it MUST be enclosed within a *Payload*
 242 *Container*.

243 If there is no application payload within the *Message Package* then a *Payload Container* MUST
 244 NOT be present.

245 The contents of each *Payload Container* MUST be identified by the ebXML Message **Manifest**
 246 element within the *SOAP Body* (see section 8.10).

247 The ebXML Message Service Specification makes no provision, nor limits in any way, the
 248 structure or content of application payloads. Payloads MAY be a simple-plain-text object or
 249 complex nested multipart objects. The specification of the structure and composition of payload
 250 objects is the prerogative of the organization that defines the business process or information
 251 exchange that uses the *ebXML Message Service*.

252 **7.4.1 Example of a Payload Container**

253 The following fragment represents an example of a *Payload Container* and a payload:

```

254 Content-ID: <domainname.example.com> ----- ebXML MIME
255 Content-Type: application/xml -----
256
257
258 <Invoice> ----- Payload Container
259   <Invoicedata>
260     ...
261   </Invoicedata>
262 </Invoice> ----- Payload
    
```

263

264 **7.5 Additional MIME Parameters**

265 Any MIME part described by this specification MAY contain additional MIME headers in
 266 conformance with the [RFC2045] specification. Implementations MAY ignore any MIME header
 267 not defined in this specification. Implementations MUST ignore any MIME header that they do not
 268 recognize.

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

271 **7.6 Reporting MIME Errors**

272 If a MIME error is detected in the *Message Package* then it MUST be reported as specified in
 273 [SOAP].

274 8 ebXML SOAP Extensions

275 The ebXML Message Service Specification defines a set of namespace-qualified SOAP **Header**
 276 and **Body** element extensions within the SOAP Envelope. In general, separate ebXML SOAP
 277 extension elements are used where:

- 278 • different software components are likely to be used to generate ebXML SOAP extension
 279 elements,
- 280 • an ebXML SOAP extension element is not always present or,
- 281 • the data contained in the ebXML SOAP extension element MAY be digitally signed
 282 separately from the other ebXML SOAP extension elements.

283 8.1 XML Prolog

284 The SOAP Message's XML Prolog, if present, MAY contain an XML declaration. This
 285 specification has defined no additional comments or processing instructions that may appear in
 286 the XML prolog. For example:

```
287 <?xml version="1.0" encoding="UTF-8"?>  
288 <SOAP-ENV:Envelope>...</SOAP-ENV:Envelope>
```

290 8.1.1 XML Declaration

291 The XML declaration MAY be present in a SOAP Message. If present, it MUST contain the
 292 version specification required by the XML Recommendation [XML]: version='1.0' and MAY
 293 contain an encoding declaration. The semantics described below MUST be implemented by a
 294 compliant *ebXML Message Service*.

295 8.1.2 Encoding Declaration

296 If both the encoding declaration and the *Header Container* MIME charset are present, the XML
 297 prolog for the SOAP Message SHALL contain the encoding declaration that SHALL be equivalent
 298 to the charset attribute of the MIME Content-Type of the *Header Container* (see section 7.3).

299 If provided, the encoding declaration MUST NOT contain a value conflicting with the encoding
 300 used when creating the SOAP Message. It is RECOMMENDED that UTF-8 be used when
 301 encoding the SOAP Message.

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

305 Note: the encoding declaration is not required in an XML document according to the XML version 1.0
 306 specification [XML].

307 For example:

```
308 Content-Type: text/xml; charset="UTF-8"  
309 <?xml version="1.0" encoding="UTF-8"?>
```

310 8.2 ebXML SOAP Envelope Extensions

311 The ebXML Message Service Specification does not extend the SOAP **Envelope** element.
 312 However, all ebXML SOAP extension element content defined in this specification MUST be
 313 namespace qualified to the ebXML Message Header namespace as defined in section 8.2.1.
 314 Namespace declarations (*xmlns* pseudo attribute) for the ebXML SOAP extensions MAY be
 315 included in the SOAP **Envelope** element, in the SOAP **Header** and **Body** elements, or directly in
 316 each of the ebXML SOAP extension elements. It is RECOMMENDED that the ebXML Message
 317 Header namespace declaration be included in the SOAP **Envelope**.

318 8.2.1 Namespace pseudo attribute

319 The ebXML Message Header namespace declaration (*xmlns* pseudo attribute) (see [XML
320 Namespace]) has a REQUIRED value of "http://www.ebxml.org/namespaces/messageHeader".

```
321
322 <SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
323   xmlns:eb="http://www.ebxml.org/namespaces/messageHeader">
324 ...
325 </SOAP-ENV:Envelope>
```

326 8.2.2 ebXML SOAP Extensions

327 An ebXML Message extends the *SOAP Message* with the following principal extension elements:

- 328 • SOAP *Header* extensions:
 - 329 - **MessageHeader** – a REQUIRED element that contains routing information for the
330 message (To/From, etc.) as well as other context information about the message
 - 331 - **TraceHeaderList** – an element that contains entries that identifies the Message
332 Service Handler(s) that sent and should receive the message. This element MAY be
333 omitted.
 - 334 - **ErrorList** – an element that contains a list of the errors that are being reported
335 against a previous message. The **ErrorList** element is only used if reporting an
336 error on a previous message.
 - 337 - **Signature** – an element that contains a digital signature that conforms to [XMLDSIG]
338 that signs data associated with the message
 - 339 - **Via** – an element that is used to convey information to the next ebXML Message
340 Service Handler that receives the message.
- 341 • SOAP *Body* extensions:
 - 342 - **Manifest** – an element that points to any data present either in the *Payload*
343 *Container* or elsewhere, e.g. on the web
 - 344 - **Acknowledgment** – an element that is used by a receiving MSH to acknowledge to
345 the sending MSH that a previous message has been received
 - 346 - **StatusData** – an element that is used by a MSH when responding to a request on
347 the status of a message that was previously received

348 8.2.3 #wildcard element content

349 Some ebXML SOAP Extension elements allow for foreign namespace-qualified element content
350 to be added to provide for extensibility. The Extension element content MUST be namespace-
351 qualified in accordance with [XMLNamespaces] and MUST belong to a foreign namespace. A
352 foreign namespace is one that is NOT <http://www.ebxml.org/namespaces/messageHeader>.

353 Any foreign namespace-qualified element added SHOULD include the SOAP *mustUnderstand*
354 attribute. If the SOAP *mustUnderstand* attribute is NOT present, the default value implied is '0'
355 (false). If an implementation of the MSH does not recognize the namespace of the element and
356 the value of the SOAP *mustUnderstand* attribute is '1' (true), the MSH SHALL report an error
357 (see section 11) with *errorCode* set to **NotSupported** and *severity* set to **error**. If the value of
358 the *mustUnderstand* attribute is '0' or if the *mustUnderstand* attribute is not present, then an
359 implementation of the MSH MAY ignore the namespace-qualified element and its content.

360 8.3 SOAP Header element

361 The SOAP *Header* element is the first child element of the SOAP *Envelope* element. It MUST
362 have a namespace qualifier that matches the SOAP *Envelope* namespace declaration for the
363 namespace "<http://schemas.xmlsoap.org/soap/envelope/>". For example:

```
364
365 <SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/" ...>
366   <SOAP-ENV:Header>...</SOAP-ENV:Header>
```

```

367 <SOAP-ENV:Body>...</SOAP-ENV:Body>
368 </SOAP-ENV:Envelope>

```

369 The SOAP **Header** element contains the ebXML SOAP **Header** extension element content
 370 identified above and described in the following sections.

371 8.4 MessageHeader element

372 The **MessageHeader** element is REQUIRED in all ebXML Messages. It MUST be present as a
 373 child element of the SOAP **Header** element.

374 The **MessageHeader** element is a composite element comprised of the following ten subordinate
 375 elements:

- 376 • **From**
- 377 • **To**
- 378 • **CPAId**
- 379 • **ConversationId**
- 380 • **Service**
- 381 • **Action**
- 382 • **MessageData**
- 383 • **QualityOfServiceInfo**
- 384 • **SequenceNumber**
- 385 • **Description**

386 The **MessageHeader** element has two REQUIRED attributes as follows:

- 387 • SOAP **mustUnderstand**
- 388 • **version**

389 8.4.1 From and To elements

390 The REQUIRED **From** element identifies the *Party* that originated the message. The REQUIRED
 391 **To** element identifies the *Party* that is the intended recipient of the message. Both **To** and **From**
 392 can be logical identifiers such as a DUNS number, or identifiers that also imply a physical location
 393 such as an eMail address.

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

395 The **PartyId** element has a single attribute, **type** and content that is a string value. The **type**
 396 attribute indicates the domain of names to which the string in the content of the **PartyId** element
 397 belongs. The value of the **type** attribute MUST be mutually agreed and understood by each of the
 398 *Parties*. It is RECOMMENDED that the value of the **type** attribute be a URI.

399 If the **PartyId type** attribute is not present, the content of the **PartyId** element MUST be a URI
 400 [RFC2396], otherwise the receiving MSH SHOULD report an error (see section 11) with
 401 **errorCode** set to **Inconsistent** and **severity** set to **Error**. It is strongly RECOMMENDED that the
 402 content of the **PartyID** element be a URI.

403 The following fragment demonstrates usage of the **From** and **To** elements.

```

404 <eb:From>
405   <eb:PartyId type="urn:duns.com">1234567890123</eb:PartyId>
406 </eb:From>
407 <eb:To>
408   <eb:PartyId>smtp:joe@example.com</eb:PartyId>
409 </eb:To>

```

410 8.4.2 CPAId element

411 The REQUIRED **CPAId** element is a string that identifies the parameters governing the exchange
 412 of messages between the parties. The recipient of a message MUST be able to resolve the
 413 **CPAId** to an individual set of parameters, taking into account the sender of the message.

414 The value of a **CPAId** element MUST be unique within a namespace that is mutually agreed by
415 the two parties. This could be a concatenation of the **From** and **To PartyId** values, a URI that is
416 prefixed with the Internet domain name of one of the parties, or a namespace offered and
417 managed by some other naming or registry service. It is RECOMMENDED that the **CPAId** be a
418 URI.

419 The **CPAId** MAY reference an instance of a CPA as defined in the ebXML Collaboration Protocol
420 Profile and Agreement Specification [EBXMLTP]. An example of the **CPAId** element follows:

```
421 <eb:CPAId>http://example.com/cpas/ourcpawithyou.xml</eb:CPAId>
```

422 If the parties are operating under a CPA, then the reliable messaging parameters are determined
423 by the appropriate elements from that CPA, as identified by the **CPAId** element.

424 If a receiver determines that a message is in conflict with the CPA, the appropriate handling of
425 this conflict is undefined by this specification. Therefore, senders SHOULD NOT generate such
426 messages unless they have prior knowledge of the receiver's capability to deal with this conflict.

427 If a receiver chooses to generate an error as a result of a detected inconsistency, then it MUST
428 report it with an **errorCode** of **Inconsistent** and a **severity** of **Error**. If it chooses to generate an
429 error because the **CPAId** is not recognized, then it MUST report it with an **errorCode** of
430 **NotRecognized** and a **severity** of **Error**.

431 8.4.3 ConversationId element

432 The REQUIRED **ConversationId** element is a string identifying the set of related messages that
433 make up a conversation between two *Parties*. It MUST be unique within the **From** and **To Party**
434 pair. The *Party* initiating a conversation determines the value of the **ConversationId** element
435 that SHALL be reflected in all messages pertaining to that conversation.

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

439 The value used for a **ConversationId** is implementation dependent. An example of the
440 **ConversationId** element follows:

```
441 <eb:ConversationId>20001209-133003-28572</eb:ConversationId>
```

442 Note: implementations are free to choose how they will identify and store conversational state related to a
443 specific conversation. Implementations SHOULD provide a facility for mapping between their
444 identification schema and a **ConversationId** generated by another implementation.

445 8.4.4 Service element

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

- 448 • a standards organization, or
- 449 • an individual or enterprise

450 Note: in the context of an ebXML Business Process model, a **Service** element identifies a Business
451 Transaction.

452 An example of the **Service** element follows:

```
453 <eb:Service>urn:services:OrderProcessing</eb:Service>
```

454 Note: URIs in the **Service** element that start with the namespace: **uri:www.ebxml.org/messageService/** are
455 reserved for use by this specification.

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

457 **8.4.4.1 type attribute**

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

461 If the **type** attribute is not present, the content of the **Service** element MUST be a URI
 462 [RFC2396]. If it is not a URI then report an error with an **errorCode** of **Inconsistent** and a
 463 **severity** of **Error** (see section 11).

464 **8.4.5 Action element**

465 The REQUIRED **Action** element identifies a process within a **Service** that processes the
 466 Message. **Action** SHALL be unique within the **Service** in which it is defined. An example of the
 467 **Action** element follows:

```
468 <eb:Action>NewOrder</eb:Action>
```

469 **8.4.6 MessageData element**

470 The REQUIRED **MessageData** element provides a means of uniquely identifying an ebXML
 471 Message. It contains the following four subordinate elements:

- 472 • **MessageId**
- 473 • **Timestamp**
- 474 • **RefToMessageId**
- 475 • **TimeToLive**

476 The following fragment demonstrates the structure of the **MessageData** element:

```
477 eb:MessageData>
478     <eb:MessageId>example.com.20001209-133003-28572</eb:MessageId>
479
480 <eb:RefToMessageId>example.com.20001209-133003-28571</eb:RefToMessageId>
481     <eb:Timestamp>20010215111212Z</Timestamp>
482 </eb:MessageData>
```

483 **8.4.6.1 MessageId element**

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

487 **8.4.6.2 Timestamp element**

488 The **Timestamp** is a value representing the time that the message header was created
 489 conforming to an [XMLSchema] `timeInstant`.

490 **8.4.6.3 RefToMessageId element**

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

494 For Error messages, the **RefToMessageId** element is REQUIRED and its value MUST be the
 495 **MessageId** value of the *message in error* (as defined in section 11).

496 For Acknowledgment Messages, the **RefToMessageId** element is REQUIRED, and its value
 497 MUST be the **MessageId** value of the ebXML Message being acknowledged. See also sections
 498 8.11.6 and 10.

499 For Message Status Response Messages, the **RefToMessageId** contains the **MessageId** of the
 500 message whose status is being reported.

501 **8.4.6.4 TimeToLive element**

502 The **TimeToLive** element indicates the time by which a message should be delivered to and
 503 processed by the *To Party*. The **TimeToLive** element is discussed under Reliable Messaging in
 504 section 10.

505 **8.4.7 QualityOfServiceInfo element**

506 The **QualityOfServiceInfo** element identifies the quality of service with which the message is
 507 delivered. This element has three attributes:

- 508 • **deliverySemantics**
- 509 • **messageOrderSemantics**
- 510 • **deliveryReceiptRequested**

511 The **QualityOfServiceInfo** element SHOULD be present if any of the attributes within the
 512 element need to be set to their non-default value. The **deliverySemantics** attribute supports
 513 Reliable Messaging and is discussed in detail in section 10. The **deliverySemantics** attribute
 514 indicates whether or not a message is sent reliably. See section 10.2.1 for more details."

515 **8.4.7.1 deliveryReceiptRequested attribute**

516 The **deliveryReceiptRequested** attribute is used by a *From Party* to indicate whether a message
 517 received by the *To Party* should result in the *To Party* returning an acknowledgment message
 518 containing an **Acknowledgment** element with a **type** of **deliveryReceipt**.

519 The **deliveryReceiptRequested** element indicates that the *To Party* has received the message.
 520 This is separate from a Reliable Messaging acknowledgment message which only indicates that
 521 a receiving MSH has successfully received a message.

522 Before setting the value of **deliveryReceiptRequested**, the *From Party* SHOULD check if the *To*
 523 *Party* supports Delivery Receipts of the type requested (see also [EBXMLTP]).

524 Valid values for **deliveryReceiptRequested** are:

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

528 The default value for **deliveryReceiptRequested** is **None**.

529 When a *To Party* receives a message with **deliveryReceiptRequested** attribute set to **Signed** or
 530 **Unsigned** then it should verify that it is able to support the type of Delivery Receipt requested.

531 If the *To Party* can produce the Delivery Receipt of the type requested, then it MUST return to the
 532 *From Party* a message containing an **Acknowledgment** element with the value of the **type**
 533 attribute set to **DeliveryReceipt**.

534 If the *To Party* cannot return a Delivery Receipt of the type requested then it MUST report the
 535 error to the *From Party* using an **errorCode** of **NotSupported** and a **severity** of **Error**.

536 An example of **deliveryReceiptRequested** follows:

```
537 <eb:QualityOfServiceInfo eb:deliverySemantics="OnceAndOnlyOnce"
538     eb:messageOrderSemantics="Guaranteed"
539     eb:deliveryReceiptRequested="Unsigned" />
```

540 **8.4.7.2 messageOrderSemantics attribute**

541 The **messageOrderSemantics** attribute is used to indicate whether the message is passed to
 542 the receiving application in the order the sending application specified. Valid Values are:

- 543 • **Guaranteed.** The messages are passed to the receiving application in the order that the
- 544 sending application specified.
- 545 • **NotGuaranteed** The messages may be passed to the receiving application in different
- 546 order from the order the sending application specified.

547 The default value for **messageOrderSemantics** is specified in the CPA or in **MessageHeader**. If

548 a value is not specified, the default value is **NotGuaranteed**.

549 If **messageOrderSemantics** is set to **Guaranteed**, the *To Party* MSH MAY correct invalid order

550 of messages using the value of **SequenceNumber** in the conversation specified by the

551 **ConversationId**. The **Guaranteed** semantics can be set only when **deliverySemantics** is

552 **OnceAndOnlyOnce**. If **deliverySemantics** is not **OnceAndOnlyOnce** then report the error to

553 the *From Party* with an **errorCode** of **Inconsistent** and a **severity** of **Error** (see sections 10 and

554 11).

555 If **messageOrderSemantics** is set to **NotGuaranteed**, then the *To Party* MSH does not need to

556 correct invalid order of messages.

557 If the *To Party* is unable to support the type of **messageOrderSemantics** requested, then the *To*

558 *Party* MUST report the error to the *From Party* using an **errorCode** of **NotSupported** and a

559 **severity** of **Error**. A sample of **messageOrderSemantics** follows.

560

561

562

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

563 8.4.8 SequenceNumber element

564 The **SequenceNumber** element indicates the sequence in which messages MUST be processed

565 by a receiving MSH. The **SequenceNumber** is unique within the **ConversationId** and MSH. The

566 *From Party* MSH and the *To Party* MSH each set an independent **SequenceNumber** as the

567 sending MSH within the **ConversationID**. It is set to zero on the first message from that MSH

568 for a conversation and then incremented by one for each subsequent message sent.

569 The **SequenceNumber** element MUST appear only when **deliverySemantics** is

570 **OnceAndOnlyOnce**. If it does not meet this criterion, then there is an error that MUST be

571 reported to the *From Party* MSH with an **errorCode** of **Inconsistent** and a **severity** of **Error**.

572 An MSH that receives a message with a **SequenceNumber** element MUST NOT pass the

573 message to an application as long as the storage required to save out-of-sequence messages is

574 within the implementation defined limits and until all the messages with lower

575 **SequenceNumbers** have been received and passed to the application.

576 If the implementation defined limit for saved out-of-sequence messages is reached, then the

577 receiving MSH MUST indicate a delivery failure to the sending MSH with **errorCode** set to

578 **DeliveryFailure** and **severity** set to **Error** (see section 11).

579 The **SequenceNumber** element is an integer value that is incremented by the sending MSH (e.g.

580 0, 1, 2, 3, 4...) for each application-prepared message sent by that MSH within the

581 **ConversationId**. The next value of 99999999 in the increment is "0". The value of

582 **SequenceNumber** consists of ASCII numerals in the range 0-99999999. In following cases,

583 **SequenceNumber** takes the value "0":

- 584 1) First message from the sending MSH within the conversation
- 585 2) First message after resetting **SequenceNumber** information by the sending MSH
- 586 3) First message after wraparound (next value after 99999999)

587 The **SequenceNumber** element has a single attribute, **status**. This attribute is an enumeration,

588 which SHALL have one of the following values:

- 589 • **Reset** – the **SequenceNumber** is reset as shown in 1 or 2 above
- 590 • **Continue** – the **SequenceNumber** continues sequentially (including 3 above)

591 When the **SequenceNumber** is set to "0" because of 1 or 2 above, the sending MSH MUST set
 592 the **status** attribute of the message to **Reset**. In all other cases, including 3 above, the **status**
 593 attribute MUST be set to **Continue**.

594 A sending MSH MUST wait before resetting the **SequenceNumber** of a conversation until it has
 595 received all of the *Acknowledgement Messages* for Messages previously sent for the
 596 conversation. Only when all the sent Messages are acknowledged, can the sending MSH reset
 597 the **SequenceNumber**. An example of **SequenceNumber** follows.

```
598 <eb:SequenceNumber status="Reset">0</eb:SequenceNumber>
```

600 8.4.9 Description element

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

606 8.4.10 version attribute

607 The REQUIRED **version** attribute indicates the version of the *ebXML Message Service Header*
 608 Specification to which the ebXML SOAP extensions conform. Its purpose is to provide future
 609 versioning capabilities. The value of the **version** attribute MUST be "0.98b". Future versions of
 610 this specification SHALL require other values of this attribute. The **version** attribute MUST be
 611 namespace qualified for the ebXML Message Header namespace defined above in section 8.2.1.

612 8.4.11 SOAP mustUnderstand attribute

613 The REQUIRED SOAP **mustUnderstand** attribute, namespace qualified to the SOAP
 614 namespace (<http://schemas.xmlsoap.org/soap/envelope/>), indicates that the contents of the
 615 **MessageHeader** element MUST be understood by a receiving process or else the message
 616 MUST be rejected in accordance with [SOAP]. This attribute MUST have a value of '1' (true).

617 8.4.12 MessageHeader sample

618 The following fragment demonstrates the structure of the **MessageHeader** element within the
 619 SOAP Header:

```
620 <eb:MessageHeader id="..." eb:version="98.0" SOAP-ENV:mustUnderstand="1">
621   <eb:From>uri:example.com</eb:From>
622   <eb:To type="someType">QRS543</eb:To>
623   <eb:CPAId>http://www.ebxml.org/cpa/123456</eb:CPAId>
624   <eb:ConversationId>987654321</eb:ConversationId>
625   <eb:Service type="myservicetypes">QuoteToCollect</eb:Service>
626   <eb:Action>NewPurchaseOrder</eb:Action>
627   <eb:MessageData>
628     <eb:MessageId>mid:UUID-2</eb:MessageId>
629     <eb:Timestamp>20000725T121905.000Z</eb:Timestamp>
630     <eb:RefToMessageId>mid:UUID-1</eb:RefToMessageId>
631   </eb:MessageData>
632   <eb:QualityOfServiceInfo
633     deliverySemantics="OnceAndOnlyOnce"
634     deliveryReceiptRequested="Signed"/>
635 </eb:MessageHeader>
```

637 8.5 TraceHeaderList element

638 A **TraceHeaderList** element consists of one or more **TraceHeader** elements. Exactly one
 639 **TraceHeader** is appended to the **TraceHeaderList** following any pre-existing **TraceHeader**
 640 before transmission of a message over a data communication protocol.

641 The **TraceHeaderList** element MAY be omitted from the header if:

- 642 • the message is being sent over a single hop (see section 8.5.4), and

- 643 • the message is not being sent reliably (see section 10)

644 The **TraceHeaderList** element has two REQUIRED attributes as follows:

- 645 • SOAP **mustUnderstand**
646 • **version**

647 **8.5.1 SOAP mustUnderstand attribute**

648 The REQUIRED SOAP **mustUnderstand** attribute, namespace qualified to the SOAP
649 namespace (<http://schemas.xmlsoap.org/soap/envelope/>), indicates that the contents of the
650 **TraceHeaderList** element MUST be understood by a receiving process or else the message
651 MUST be rejected in accordance with [SOAP]. This attribute MUST have a value of '1' (true).

652 **8.5.2 version attribute**

653 The REQUIRED **version** attribute indicates the version of the *ebXML Message Service Header*
654 Specification to which the *ebXML SOAP Header* extensions conform. Its purpose is to provide
655 future versioning capabilities. The value of the **version** attribute MUST be "0.98b". Future
656 versions of this specification SHALL require other values of this attribute. The version attribute
657 MUST be namespace qualified for the *ebXML Message Header* namespace defined above.

658 **8.5.3 TraceHeader element**

659 The **TraceHeader** element contains information about a single transmission of a message
660 between two instances of an MSH. If a message traverses multiple hops by passing through one
661 or more intermediate MSH nodes as it travels between the *From Party* MSH and the *To Party*
662 MSH, then each transmission over each successive "hop" results in the addition of a new
663 **TraceHeader** element by the sending MSH.

664 The **TraceHeader** element is a composite element comprised of the following subordinate
665 elements:

- 666 • **SenderURI**
667 • **ReceiverURI**
668 • **Timestamp**
669 • **#wildcard**

670 **8.5.3.1 SenderURI element**

671 This element contains the URI of the Sender's Message Service Handler. Unless there is
672 another URI identified within the CPA or in **MessageHeader** (section 8.4.2), the recipient of the
673 message uses the URI to send a message, when required that:

- 674 • responds to an earlier message
675 • acknowledges an earlier message
676 • reports an error in an earlier message.

677 **8.5.3.2 ReceiverURI element**

678 This element contains the URI of the Receiver's Message Service Handler. It is the URI to which
679 the Sender sends the message.

680 **8.5.3.3 Timestamp element**

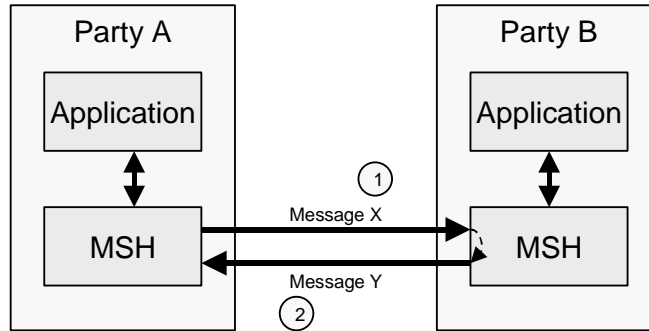
681 The **Timestamp** element is the time the individual **TraceHeader** was created. It is in the same
682 format as in the **Timestamp** element in the **MessageData** element (section 8.4.6.2).

683 **8.5.3.4 #wildcard element**

684 Refer to section 8.2.3 for discussion of #wildcard element handling.

685 **8.5.4 Single Hop TraceHeader Sample**

686 A single hop message is illustrated by the diagram below.



687

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

689 The content of the corresponding messages could include:

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

690

691

692

693

694

695

696

697

698

699

700

701

702

703

704

705

706

707

708

709

710

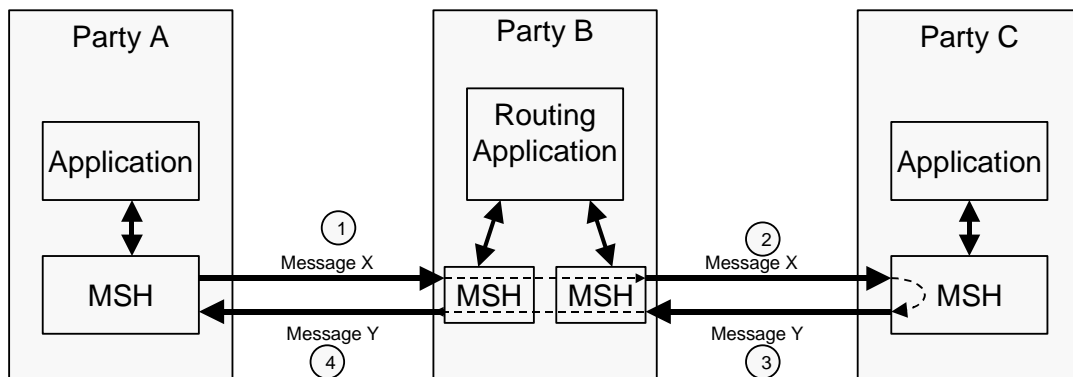
```

<eb:MessageHeader id="..." eb:version="98.0" SOAP-ENV:mustUnderstand="1">
  <eb:From>urn:myscheme.com:id:PartyA-id</eb:From>
  <eb:To>urn:myscheme.com:id:PartyB-id</eb:To>
  <eb:ConversationId>219cdj89dj2398djfn</eb:ConversationId>
  ...
  <eb:MessageData>
    <eb:MessageId>29dmridj103kvna</eb:MessageId>
    ...
  </eb:MessageData>
  ...
</eb:MessageHeader>

<eb:TraceHeaderList id="..." eb:version="98.0" SOAP-ENV:mustUnderstand="1">
  <eb:TraceHeader>
    <eb:SenderURI>url:PartyA.com/PartyAMsh</eb:SenderURI>
    <eb:ReceiverURI>url:PartyB.com/PartyBMsh</eb:ReceiverURI>
    <eb:Timestamp>20001216T21:19:35.145Z-8</eb:Timestamp>
  </eb:TraceHeader>
</eb:TraceHeaderList>
    
```

711 **8.5.5 Multi-hop TraceHeader Sample**

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



714

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

716 The content of the corresponding messages could include:

717 • **Transmission 1 - Message X From Party A To Party B**

```

718 <eb:MessageHeader id="..." eb:version="98.0" SOAP-ENV:mustUnderstand="1">
719   <eb:From>urn:myscheme.com:id:PartyA-id</eb:From>
720   <eb:To>urn:myscheme.com:id:PartyC-id</eb:To>
721   <eb:ConversationId>219cdj89dj2398djfjn</eb:ConversationId>
722   ...
723   <eb:MessageData>
724     <eb:MessageId>29dmridj103kvna</eb:MessageId>
725     ...
726   </eb:MessageData>
727   ...
728 </eb:MessageHeader>
729
730 <eb:TraceHeaderList id="..." eb:version="98.0" SOAP-ENV:mustUnderstand="1">
731   <eb:TraceHeader>
732     <eb:SenderURI>url:PartyA.com/PartyAMsh</eb:SenderURI>
733     <eb:ReceiverURI>url:PartyB.com/PartyBMsh</eb:ReceiverURI>
734     <eb:Timestamp>20001216T21:19:35.145Z-8</eb:Timestamp>
735   </eb:TraceHeader>
736 </eb:TraceHeaderList>

```

737 • **Transmission 2 - Message X From Party B To Party C**

```

738 <eb:MessageHeader id="..." eb:version="98.0" SOAP-ENV:mustUnderstand="1">
739   <eb:From>urn:myscheme.com:id:PartyA-id</eb:From>
740   <eb:To>urn:myscheme.com:id:PartyC-id</eb:To>
741   <eb:ConversationId>219cdj89dj2398djfjn</eb:ConversationId>
742   ...
743   <eb:MessageData>
744     <eb:MessageId>29dmridj103kvna</eb:MessageId>
745     ...
746   </eb:MessageData>
747   ...
748 </eb:MessageHeader>
749
750 <eb:TraceHeaderList id="..." eb:version="98.0" SOAP-ENV:mustUnderstand="1">
751   <eb:TraceHeader>
752     <eb:SenderURI>url:PartyA.com/PartyAMsh</eb:SenderURI>
753     <eb:ReceiverURI>url:PartyB.com/PartyBMsh</eb:ReceiverURI>
754     <eb:Timestamp>20001216T21:19:35.145Z-8</eb:Timestamp>
755   </eb:TraceHeader>
756   <eb:TraceHeader>
757     <eb:SenderURI>url:PartyB.com/PartyAMsh</eb:SenderURI>
758     <eb:ReceiverURI>url:PartyC.com/PartyBMsh</eb:ReceiverURI>
759     <eb:Timestamp>20001216T21:19:45.483Z-6</eb:Timestamp>
760   </eb:TraceHeader>
761 </eb:TraceHeaderList>

```

762 8.6 Via element

763 The **Via** element is a *SOAP Header* that is used to convey information to the next ebXML
764 Message Service Handler (MSH) that receives the message.

765 Note: this MSH can be a MSH operated by an intermediary or by the *To Party*. In particular, the **Via**
766 element is used to hold data that can vary from one hop to another.

767 The **Via** element **MUST** contain the following attributes:

- 768 • SOAP **mustUnderstand** attribute with a value of '1'
- 769 • SOAP **actor** attribute with the value "<http://schemas.xmlsoap.org/soap/actor/next>"
- 770 • a **version** attribute

771 The **Via** element **MUST** also contain one or more of the following elements or attributes:

- 772 • **syncReply** attribute
- 773 • **reliableMessagingMethod** attribute
- 774 • **ackRequested** attribute
- 775 • **CPAId** element

776 The **Via** element MAY also contain the following elements:

- 777 • **Service** element
- 778 • **Action** element

779 8.6.1 SOAP mustUnderstand attribute

780 The REQUIRED SOAP **mustUnderstand** attribute, namespace qualified to the SOAP envelope
781 namespace (<http://schemas.xmlsoap.org/soap/envelope/>), indicates that the contents of the **Via**
782 element MUST be understood by a receiving process or else the message MUST be rejected in
783 accordance with [SOAP]. This attribute MUST have a value of '1' (true). In accordance with the
784 [SOAP] specification, a receiving *ebXML Message Service* implementation that does not provide
785 support for the **Via** element MUST respond with a SOAP Fault with a **faultCode** of
786 "MustUnderstand".

787 8.6.2 SOAP actor attribute

788 The **Via** element MUST contain a SOAP **actor** attribute with the value
789 <http://schemas.xmlsoap.org/soap/actor/next> and be interpreted and processed as defined in the
790 [SOAP] specification. This means that the **Via** element MUST be processed by the SOAP
791 processor that receives the message and SHOULD NOT be forwarded to the next SOAP
792 processor.

793 8.6.3 version attribute

794 The REQUIRED **version** attribute indicates the version of the *ebXML Message Service* Header
795 Specification to which the ebXML SOAP **Header** extensions conform. Its purpose is to provide
796 future versioning capabilities. The value of the **version** attribute MUST be "0.98b". Future
797 versions of this specification SHALL require other values of this attribute. The **version** attribute
798 MUST be namespace qualified for the ebXML Message Header namespace defined above.

799 8.6.4 syncReply attribute

800 The **syncReply** attribute is used only if the data communication protocol is *synchronous* (e.g.
801 HTTP). It is an [XML Schema] boolean. If the communication protocol is not synchronous, then
802 the value of **syncReply** is ignored. If the **syncReply** attribute is not present, it is semantically
803 equivalent to its presence with a value of "false". If the **syncReply** attribute is present with a
804 value of **true**, the MSH must get data from the application or business process and return it in the
805 payload of the synchronous response. See also the description of **syncReply** in the [EBXMLTP]
806 specification.

807 8.6.5 reliableMessagingMethod attribute

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

- 810 • **ebXML**
- 811 • **Transport**

812 The default implied value for this attribute is **ebXML**

813 8.6.6 ackRequested attribute

814 The **ackRequested** attribute is an enumeration that SHALL have one of the following values:

- 815 • **Signed**
- 816 • **UnSigned**
- 817 • **None**

818 The default implied value for this attribute is **None**. This attribute is used to indicate to the
819 receiving MSH whether an acknowledgment message is expected, and if so, whether the
820 acknowledgment message should be signed by the receiving MSH. Refer to section 10.2.5 for a
821 complete discussion as to the use of this attribute.

822 8.6.7 CPAId element

823 The **CPAId** element is a string that identifies the parameters that govern the exchange of
824 messages between two MSH instances. It has the same meaning as the **CPAId** in the
825 **MessageHeader** except that the parameters identified by the **CPAId** apply just to the exchange
826 of messages between the two MSH instances rather than between the *Parties* identified in the **To**
827 and **From** elements of the **MessageHeader** (section 8.4.2). This allows different parameters,
828 transport protocols, etc, to be used on different hops when a message is passed through
829 intermediaries.

830 If the **CPAId** element is present, the identified parameter values SHOULD be used instead of the
831 values identified by the **CPAId** in the **MessageHeader** element.

832 8.6.8 Service and Action elements

833 The **Service** and **Action** elements have the same meaning as the **Service** and **Action** elements
834 in the **MessageHeader** element (see sections 8.4.4 and 8.4.5) except that they are interpreted
835 and acted on by the next MSH whether or not the MSH is operated by the *To Party*.

836 The designer of the service or business process that is using the *ebXML Message Service*
837 defines the values used for **Service** and **Action**.

838 The **Service** and **Action** elements are OPTIONAL. However, if the **Service** element is present
839 then the **Action** element MUST also be present and vice versa.

840 8.6.9 Sample Via element

841 The following is a sample **Via** element.

```
842  
843 <eb:Via SOAP-ENV:mustUnderstand="1"  
844   SOAP-ENV:actor="http://schemas.xmlsoap.org/soap/actor/next "  
845   eb:version="98.0"  
846   eb:syncReply="false">  
847   <eb:CPAId>yaddaydda</eb:CPAId>  
848   <eb:Service>Proxy</eb:Service>  
849   <eb:Action>LogActivity</eb:Action>  
850 </eb:Via>
```

851 8.7 ErrorList element

852 The existence of an **ErrorList** element within the SOAP **Header** element indicates that the
853 message that is identified by the **RefToMessageId** in the **MessageHeader** element has an error.

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

- 855 • **id** attribute
- 856 • SOAP **mustUnderstand** attribute
- 857 • **version** attribute
- 858 • **highestSeverity** attribute

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

860 8.7.1 id attribute

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

862 8.7.2 SOAP mustUnderstand attribute

863 The REQUIRED SOAP **mustUnderstand** attribute, namespace qualified to the SOAP
864 namespace (<http://schemas.xmlsoap.org/soap/envelope/>), indicates that the contents of the
865 **ErrorList** element MUST be understood by a receiving process or else the message MUST be
866 rejected in accordance with [SOAP]. This attribute MUST have a value of '1' (true).

867 **8.7.3 version attribute**

868 The REQUIRED **version** attribute indicates the version of the *ebXML Message Service Header*
869 *Specification* to which the *ebXML SOAP Header extensions* conform. Its purpose is to provide for
870 future versioning capabilities. The value of the **version** attribute MUST be "0.98b". Future
871 versions of this specification SHALL require other values of this attribute. The version attribute
872 MUST be namespace qualified for the *ebXML Message Header namespace* defined above.

873 **8.7.4 highestSeverity attribute**

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

877 **8.7.5 Error element**

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

- 879 • **codeContext**
- 880 • **errorCode**
- 881 • **severity**
- 882 • **location**
- 883 • **xml:lang**

884 The content of the **Error** element contains an error message.

885 **8.7.5.1 codeContext attribute**

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

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

893 **8.7.5.2 errorCode attribute**

894 The REQUIRED **errorCode** attribute indicates the nature of the error in the *message in error*.
895 Valid values for the **errorCode** and a description of the code's meaning are given in sections
896 8.7.8 and 8.7.9.

897 **8.7.5.3 severity attribute**

898 The REQUIRED **severity** attribute indicates the severity of the error. Valid values are:

- 899 • **Warning** - This indicates that although there is an error, other messages in the
900 conversation will still be generated in the normal way.
- 901 • **Error** - This indicates that there is an unrecoverable error in the message and no further
902 messages will be generated as part of the conversation.

903 **8.7.5.4 location attribute**

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

905 If an error exists in an ebXML element and the element is "well formed" (see [XML]), then the
906 content of the **location** attribute MUST be an [XPointer].

907 If the error is associated with the MIME envelope that wraps the SOAP envelope and the ebXML
 908 Payload, then **location** contains the `content-id` of the MIME part that is in error, in the format
 909 `cid:23912480wsr`, where the text after the ":" is the value of the MIME part's `content-id`.

910 The **location** attribute MUST NOT be used to point to errors in payloads inside a *Payload*
 911 *Container* as the method of reporting errors in payloads is application dependent.

912 **8.7.5.5 Error element Content**

913 The content of the error message provides a narrative description of the error in the language
 914 defined by the **xml:lang** attribute. Typically, it will be the message generated by the XML parser
 915 or other software that is validating the message. This means that the content is defined by the
 916 vendor/developer of the software that generated the **Error** element.

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

918 The content of the **Error** element can be empty.

919 **8.7.6 Examples**

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

```
921 <eb:ErrorList id='3490sdo9', highestSeverity="error" eb:version="98.0"
922 SOAP-ENV:mustUnderstand="1">
923   <eb:Error errorCode='SecurityFailure' severity="Error"
924     location='URI_of_ds:Signature_goes_here' xml:lang="us-en">
925     Validation of signature failed </eb:Error>
926   <eb:Error ... </eb:Error>
927 </eb:ErrorList>
```

928 **8.7.7 errorCode values**

929 This section describes the values for the **errorCode** element (see section 8.7.5.2) used in a
 930 *message reporting an error*. They are described in a table with three headings:

- 931 • the first column contains the value to be used as an **errorCode**, e.g. **SecurityFailure**
- 932 • the second column contains a "Short Description" of the **errorCode**. Note: this narrative
 933 MUST NOT be used in the content of the **Error** element.
- 934 • the third column contains a "Long Description" that provides an explanation of the
 935 meaning of the error and provides guidance on when the particular **errorCode** should be
 936 used.

937 It is RECOMMENDED that implementers of software conforming to this specification make
 938 available to a user being informed of the error: the value of the **errorCode**, the "Short
 939 Description" and optionally the "Long Description".

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

942 **8.7.8 Reporting Errors in the ebXML Elements**

943 The following list contains error codes that can be associated with ebXML elements:
 944

Error Code	Short Description	Long Description
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 <i>ebXML Message Service</i> .
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

		<ul style="list-style-type: none"> is not supported by the <i>ebXML Message Service</i> 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 content of the Error element should be used to indicate the nature of the problem.

945 **8.7.9 Non-XML Document Errors**

946 The following are error codes that identify errors not associated with the ebXML elements:

Error Code	Short Description	Long Description
DeliveryFailure	Message Delivery Failure	A message has been received that either probably or definitely could not be sent to its next destination. Note: if <i>severity</i> 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 MessageHeader 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 content of the Error element should be used to indicate the nature of the problem.

947 **8.8 Signature element**

948 An ebXML Message may be digitally signed to provide security countermeasures. Zero or more
 949 **Signature** elements, belonging to the [XMLDSIG] defined namespace MAY be present in the
 950 *SOAP Header*. The **Signature** element MUST be namespace qualified in accordance with
 951 [XMLDSIG]. The structure and content of the **Signature** element MUST conform to the
 952 [XMLDSIG] specification. If there is more than one **Signature** element contained within the
 953 *SOAP Header*, the first MUST represent the digital signature of the ebXML Message as signed by
 954 the *From Party* MSH in conformance with section 12. Additional **Signature** elements MAY be
 955 present, but their purpose is undefined by this specification.

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

958 **8.9 SOAP Body Extensions**

959 The *SOAP Body* element is the second child element of the *SOAP Envelope* element. It MUST
 960 have a namespace qualifier that matches the *SOAP Envelope* namespace declaration for the
 961 namespace "<http://schemas.xmlsoap.org/soap/envelope/>". For example:

```
962 <SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/" ...>
963   <SOAP-ENV:Header>...</SOAP-ENV:Header>
964   <SOAP-ENV:Body>...</SOAP-ENV:Body>
965 </SOAP-ENV:Envelope>
```

966 The SOAP **Body** element contains the ebXML SOAP **Body** extension element content as
967 follows:

- 968 • **Manifest** element
- 969 • **StatusData** element
- 970 • **Acknowledgement** element

971 Each is defined in the following sections.

972 8.10 Manifest element

973 The **Manifest** element is a composite element consisting of one or more **Reference** elements.
974 Each **Reference** element identifies data associated with the message, whether included as part
975 of the message as payload document(s) contained in a *Payload Container*, or remote resources
976 accessible via a URL. It is RECOMMENDED that no payload data be present in the SOAP-
977 ENV:Body. The purpose of the **Manifest** is as follows:

- 978 • to make it easier to directly extract a particular payload associated with this ebXML
979 Message,
- 980 • to enable a MSH to check the integrity of an ebXML Message
- 981 • to allow an application to determine whether it can process the payload without having to
982 parse it.

983 The **Manifest** element is comprised of the following attributes and elements, each of which is
984 described below:

- 985 • a REQUIRED **id** attribute
- 986 • a REQUIRED SOAP **mustUnderstand** attribute
- 987 • a REQUIRED **version** attribute
- 988 • one or more **Reference** elements
- 989 • **#wildcard**

990 8.10.1 id attribute

991 The **Manifest** element MUST have an **id** attribute that is an XML ID.

992 8.10.2 SOAP mustUnderstand attribute

993 The REQUIRED SOAP **mustUnderstand** attribute, namespace qualified to the SOAP
994 namespace (<http://schemas.xmlsoap.org/soap/envelope/>), indicates that the contents of the
995 **Manifest** element MUST be understood by a receiving process or else the message MUST be
996 rejected in accordance with [SOAP]. This attribute MUST have a value of '1' (true).

997 8.10.3 version attribute

998 The REQUIRED **version** attribute indicates the version of the *ebXML Message Service Header*
999 Specification to which the ebXML SOAP **Header** extensions conform. Its purpose is to provide
1000 future versioning capabilities. The value of the **version** attribute MUST be "0.98b". Future
1001 versions of this specification SHALL require other values of this attribute. The version attribute
1002 MUST be namespace qualified for the ebXML Message Header namespace defined above.

1003 8.10.4 #wildcard element

1004 Refer to section 8.2.3 for discussion of #wildcard element handling.

1005 8.10.5 Reference element

1006 The **Reference** element is a composite element consisting of the following subordinate elements:

- 1007 • **Schema** - information about the schema(s) that define the instance document identified
- 1008 in the parent **Reference** element
- 1009 • **Description** - a textual description of the payload object referenced by the parent
- 1010 **Reference** element
- 1011 • **#wildcard** - any namespace-qualified element content belonging to a foreign namespace

1012 The **Reference** element itself is an [XLINK] simple link. XLINK is presently a Candidate
 1013 Recommendation (CR) of the W3C. It should be noted that the use of XLINK in this context is
 1014 chosen solely for the purpose of providing a concise vocabulary for describing an association.
 1015 Use of an XLINK processor or engine is NOT REQUIRED, but MAY prove useful in certain
 1016 implementations.

1017 The **Reference** element has the following attribute content in addition to the element content
 1018 described above:

- 1019 • **id** - a REQUIRED XML ID for the **Reference** element,
- 1020 • **xlink:type** - this attribute defines the element as being an XLINK simple link. It has a
 1021 fixed value of 'simple',
- 1022 • **xlink:href** - this REQUIRED attribute has a value that is the URI of the payload object
 1023 referenced. It SHALL conform to the [XLINK] specification criteria for a simple link.
- 1024 • **xlink:role** - this attribute identifies some resource that describes the payload object or its
 1025 purpose. If present, then it SHALL have a value that is a valid URI in accordance with the
 1026 [XLINK] specification,
- 1027 • Any other namespace-qualified attribute MAY be present. A receiving MSH MAY choose
 1028 to ignore any foreign namespace attributes other than those defined above.

1029 8.10.5.1 Schema element

1030 If the item being referenced has schema(s) of some kind that describe it (e.g. an XML Schema,
 1031 DTD, or a database schema), then the **Schema** element SHOULD be present as a child of the
 1032 **Reference** element. It provides a means of identifying the schema and its version defining the
 1033 payload object identified by the parent **Reference** element. The **Schema** element contains the
 1034 following attributes:

- 1035 • **location** - the REQUIRED URI of the schema
- 1036 • **version** - a version identifier of the schema

1037 8.10.5.2 Description element

1038 The **Reference** element MAY contain zero or more **Description** elements. The **Description** is a
 1039 textual description of the payload object referenced by the parent **Reference** element. The
 1040 language of the description is defined by a REQUIRED **xml:lang** attribute. The **xml:lang** attribute
 1041 MUST comply with the rules for identifying languages specified in [XML]. This element is provided
 1042 to allow a human readable description of the payload object identified by the parent **Reference**
 1043 element. If multiple **Description** elements are present, each SHOULD have a unique **xml:lang**
 1044 attribute value. An example of a **Description** element follows.

1045 `<eb:Description xml:lang="en-gb">Purchase Order for 100,000 widgets</eb:Description>`

1046 8.10.5.3 #wildcard element

1047 Refer to section 8.2.3 for discussion of #wildcard element handling.

1048 8.10.6 What References are Included in a Manifest

1049 The designer of the business process or information exchange that is using ebXML Messaging
 1050 decides what payload data is referenced by the Manifest and the values to be used for **xlink:role**.

1051 8.10.7 Manifest Validation

1052 If an *xlink:href* attribute contains a URI that is a content id (URI scheme "cid") then a MIME
 1053 part with that *content-id* MUST be present in the *Payload Container* of the message. If it is
 1054 not, then the error SHALL be reported to the *From Party* with an *errorCode* of *MimeProblem*
 1055 and a *severity* of *Error*.

1056 If an *xlink:href* attribute contains a URI that is not a content id (URI scheme "cid") and that URI
 1057 cannot be resolved, then it is an implementation decision on whether to report the error. If the
 1058 error is to be reported, then it SHALL be reported to the *From Party* with an *errorCode* of
 1059 *MimeProblem* and a *severity* of *Error*.

1060 8.10.8 Manifest sample

1061 The following fragment demonstrates a typical *Manifest* for a message with a single payload
 1062 MIME body part:

```
1063 <eb:Manifest id="Manifest" eb:version="98.0" SOAP-ENV:mustUnderstand="1">
1064   <eb:Reference id="pay01"
1065     xlink:href="cid:payload-1"
1066     xlink:role="http://regrep.org/gci/purchaseOrder">
1067     <eb:Description>Purchase Order for 100,000 widgets</eb:Description>
1068     <eb:Schema location="http://regrep.org/gci/purchaseOrder/po.xsd"
1069       version="98.0"/>
1070   </eb:Reference>
1071 </eb:Manifest>
```

1073 8.11 StatusData Element

1074 The *StatusData* element is used by one MSH to respond to a request on the status of the
 1075 processing of a message that was previously sent (see also section 9.1).

1076 The *StatusData* element consists of the following elements and attributes:

- 1077 • a REQUIRED *RefToMessageld* element
- 1078 • a *Timestamp* element
- 1079 • a REQUIRED SOAP *mustUnderstand* attribute
- 1080 • a REQUIRED *version* attribute
- 1081 • a *messageStatus* attribute

1082 8.11.1 RefToMessageld element

1083 A REQUIRED *RefToMessageld* element that contains the *Messageld* of the message whose
 1084 status is being reported.

1085 8.11.2 Timestamp element

1086 The *Timestamp* element contains the time that the message, whose status is being reported,
 1087 was received (section 8.4.6.2.). This MUST be omitted if the message whose status is being
 1088 reported is *NotRecognized* or the request was *Unauthorized*.

1089 8.11.3 SOAP mustUnderstand attribute

1090 The REQUIRED SOAP *mustUnderstand* attribute, namespace qualified to the SOAP
 1091 namespace (<http://schemas.xmlsoap.org/soap/envelope/>), indicates that the contents of the
 1092 *StatusData* element MUST be understood by a receiving process or else the message MUST be
 1093 rejected in accordance with [SOAP]. This attribute MUST have a value of '1' (true).

1094 8.11.4 version attribute

1095 The REQUIRED *version* attribute indicates the version of the *ebXML Message Service Header*
 1096 Specification to which the *ebXML SOAP Header extensions* conform. Its purpose is to provide
 1097 future versioning capabilities. The value of the *version* attribute MUST be "0.98b". Future

1098 versions of this specification SHALL require other values of this attribute. The **version** attribute
1099 MUST be namespace qualified for the ebXML Message Header namespace defined above.

1100 8.11.5 messageStatus attribute

1101 The **messageStatus** attribute identifies the status of the message that is identified by the
1102 **RefToMessageld** element. It SHALL be set to one of the following values:

- 1103 • **Unauthorized** – the Message Status Request is not authorized or accepted
- 1104 • **NotRecognized** – the message identified by the **RefToMessageld** element in the
1105 **StatusData** element is not recognized
- 1106 • **Received** – the message identified by the **RefToMessageld** element in the **StatusData**
1107 element has been received by the MSH

1108 Note: if a Message Status Request is sent after the elapsed time indicated by **persistDuration** has passed
1109 since the message being queried was sent, then the Message Status Response may indicate that the
1110 **MessageId** was **NotRecognized** as the **MessageId** is no longer in persistent storage.

1111 8.11.6 StatusData sample

1112 An example of the **StatusData** element is given below:

```
1113 <eb:StatusData SOAP-ENV:mustUnderstand="1"
1114   eb:version="98.0" eb:messageStatus="Received">
1115   <eb:RefToMessageId>323210:e52151ec74:-7ffc@xtacy</eb:RefToMessageId>
1116   <eb:Timestamp>20010309T122230.105Z</eb:Timestamp></eb:StatusData>
```

1117 8.12 Acknowledgment Element

1118 The **Acknowledgment** element is an optional element that is used by one Message Service
1119 Handler to indicate that another Message Service Handler has received a message. The
1120 **RefToMessageld** in a message containing an **Acknowledgement** element is used to identify the
1121 message being acknowledged by its **MessageId**.

1122 The **Acknowledgment** element consists of the following elements and attributes:

- 1123 • a **Timestamp** element
- 1124 • a **From** element
- 1125 • a REQUIRED SOAP **mustUnderstand** attribute
- 1126 • a REQUIRED **version** attribute
- 1127 • a **type** attribute
- 1128 • a **signed** attribute

1129 8.12.1 Timestamp element

1130 The **Timestamp** element is a value representing the time that the *message being acknowledged*
1131 was received by the Party generating the *acknowledgment message*. It must conform to an
1132 [XMLSchema] `timelInstant` (section 8.4.6.2).

1133 8.12.2 From element

1134 This is the same element as the **From** element within **MessageHeader** element (see section
1135 8.4.1). However, when used in the context of an **Acknowledgment** element, it contains the
1136 identifier of the *Party* that is generating the *acknowledgment message*.

1137 If the **From** element is omitted then the *Party* that is sending the element is identified by the **From**
1138 element in the **MessageHeader** element.

1139 8.12.3 SOAP mustUnderstand attribute

1140 The REQUIRED SOAP **mustUnderstand** attribute, namespace qualified to the SOAP
1141 namespace (<http://schemas.xmlsoap.org/soap/envelope/>), indicates that the contents of the

1142 **Acknowledgment** element MUST be understood by a receiving process or else the message
1143 MUST be rejected in accordance with [SOAP]. This attribute MUST have a value of '1' (true).

1144 8.12.4 version attribute

1145 The REQUIRED **version** attribute indicates the version of the *ebXML Message Service Header*
1146 *Specification* to which the *ebXML SOAP Header extensions* conform. Its purpose is to provide
1147 future versioning capabilities. The value of the **version** attribute MUST be "0.98b". Future
1148 versions of this specification SHALL require other values of this attribute. The version attribute
1149 MUST be namespace qualified for the *ebXML Message Header namespace* defined above.

1150 8.12.5 type attribute

1151 The **type** attribute indicates who sent the *acknowledgment message*. It MUST contain either:
1152

- **DeliveryReceipt** - indicates that the *acknowledgment message* was generated by the *To*
1153 *Party* identified by the **To** element of the *message being acknowledged*, or
- **Acknowledgement** - indicates that the *acknowledgment message* was generated by a
1154 *Party* that is not the *To Party* identified by the **To** element of the *message being*
1155 *acknowledged*. Typically this will be a *Party* that has received the message and is
1156 forwarding it to either the *To Party* or another *Party* with the intention that the message is
1157 sent to the *To Party*.
1158

1159 The default value for **type** is **DeliveryReceipt**.

1160 8.12.6 signed attribute

1161 The **signed** attribute indicates whether the *acknowledgment message* is digitally signed. It MUST
1162 contain either:

- 1163 • **true** - indicates that the *acknowledgment message* is digitally signed, or
- 1164 • **false** - indicates that the *acknowledgment message* is not digitally signed

1165 The default value for **signed** is **false**.

1166 See section 12 for details on what should be signed and how a signature that signs an
1167 *acknowledgment message* should be checked.

1168 8.12.7 Acknowledgement sample

1169 An example of the **Acknowledgement** element is given below:

```
1170 <eb:Acknowledgment SOAP-ENV:mustUnderstand="1"
1171   eb:version="0.98b" eb:type="Acknowledgment" eb:signed="false">
1172   <eb:Timestamp>20010309T122230.109Z</eb:Timestamp>
1173 </eb:Acknowledgment>
```

1174 8.13 Combining ebXML SOAP Extension Elements

1175 This section describes how the various ebXML SOAP extension elements may be used in
1176 combination.

1177 8.13.1 Manifest element

1178 The **Manifest** element MUST be present if there is any data associated with the message that is
1179 not present in the *Header Container*. This applies specifically to data in the *Payload Container* or
1180 elsewhere, e.g. on the web.

1181 8.13.2 MessageHeader element

1182 The **MessageHeader** element MUST be present in every message.

1183 8.13.3 TraceHeaderList element

1184 The **TraceHeaderList** element MAY be present in any message. It MUST be present if the
1185 message is being sent reliably (see section 10) or over multiple hops (see section 8.5.5).

1186 8.13.4 StatusData element

1187 This element MUST NOT be present with the following elements:

- 1188 • a *Manifest* element
- 1189 • an *ErrorList* element with a *highestSeverity* attribute set to *Error*

1190 8.13.5 ErrorList element

1191 If the *highestSeverity* attribute on the *ErrorList* is set to *Warning*, then this element MAY be
1192 present with any other element.

1193 If the *highestSeverity* attribute on the *ErrorList* is set to *Error*, then this element MUST NOT be
1194 present with the following:

- 1195 • a *Manifest* element
- 1196 • a *StatusData* element

1197 8.13.6 Acknowledgment element

1198 An *Acknowledgment* element MAY be present on any message.

1199 8.13.7 Signature element

1200 One or more *Signature* elements MAY be present on any message.

1201 8.13.8 Via element

1202 One-and-only-one *Via* element MAY be present in any message.

1203 9 Message Service Handler Services

1204 The Message Service Handler MAY support two services that are designed to help provide
1205 smooth operation of a Message Handling Service implementation:

- 1206 • Message Status Request
- 1207 • Message Service Handler Ping

1208 If a receiving MSH does not support the service requested, it SHOULD return a SOAP Fault with
1209 a **faultCode** of **MustUnderstand**. Each service is described below:

1210 9.1 Message Status Request Service

1211 The Message Status Request Service consists of the following:

- 1212 • A Message Status Request message containing details regarding a message previously
1213 sent is sent to a Message Service Handler (MSH)
- 1214 • The Message Service Handler receiving the request responds with a Message Status
1215 Response message.

1216 A Message Service Handler SHOULD respond to Message Status Requests that have been sent
1217 reliably (see section 10) and the **MessageId** in the **RefToMessageId** is present in *persistent*
1218 *storage* (see section 10.1.1).

1219 An implementation MAY also respond to Message Status Requests that have not been sent
1220 reliably.

1221 A Message Service also SHOULD NOT use the Message Status Request Service to implement
1222 Reliable Messaging.

1223 9.1.1 Message Status Request Message

1224 A Message Status Request message consists of an ebXML Message containing no *ebXML*
1225 *Payload* and the following elements in the ebXML Header:

- 1226 • A **MessageHeader** element
- 1227 • A **TraceHeaderList** element
- 1228 • A **Signature** element

1229 The **TraceHeaderList** and the **Signature** elements MAY be omitted (see sections 8.5 and
1230 8.13.7).

1231 The **MessageHeader** element MUST contain the following:

- 1232 • a **From** element that identifies the party that created the message status request
1233 message
- 1234 • a **To** element that identifies a party who should receive the message. If a **TraceHeader**
1235 was present on the message whose status is being checked, this MUST be the
1236 **ReceiverURI** from that message
- 1237 • a **RefToMessageId** element within the **MessageData** element containing the **MessageId**
1238 of the message whose status is being queried
- 1239 • a **Service** element that contains: **uri:www.ebxml.org/messageService**
- 1240 • an **Action** element that contains **StatusRequest**

1241 The message is then sent to the **To** party.

1242 9.1.2 Message Status Response Message

1243 Once the **To** party receives the Message Status Request message, they SHOULD generate a
1244 Message Status Response message consisting of no ebXML Payload and the following elements
1245 in the ebXML Header.

- 1246 • a **MessageHeader** element
- 1247 • a **TraceHeaderList** element
- 1248 • an **Acknowledgment** element
- 1249 • a **StatusData** element (see section 8.11)
- 1250 • a **Signature** element

1251 The **TraceHeaderList**, **Acknowledgment** and **Signature** elements MAY be omitted (see
1252 sections 8.5, 8.13.6 and 8.13.7).

1253 The **MessageHeader** element MUST contain the following:

- 1254 • a **From** element that identifies the sender of the Message Status Response message
- 1255 • a **To** element that is set to the value of the **From** element in the Message Status Request
1256 message
- 1257 • a **Service** element that contains the value: **uri:www.ebxml.org/messageService/**
- 1258 • an **Action** element that contains **StatusResponse**
- 1259 • a **RefToMessageId** that identifies the Message Status Request message.

1260 The message is then sent to the **To** party.

1261 9.1.3 Security Considerations

1262 Parties who receive a Message Status Request message SHOULD always respond to the
1263 message. However, they MAY ignore the message instead of responding with **messageStatus**
1264 set to **Unauthorized** if they consider that the sender of the message is unauthorized. The
1265 decision process that results in this course of action is implementation dependent.

1266 9.2 Message Service Handler Ping Service

1267 The Message Service Handler Ping Service enables one MSH to determine if another MSH is
1268 operating. It consists of:

- 1269 • sending a Message Service Handler Ping message to a MSH, and
- 1270 • the MSH that receives the Ping responding with a Message Service Handler Pong
1271 message.

1272 9.2.1 Message Service Handler Ping Message

1273 A Message Service Handler Ping (MSH Ping) message consists of an ebXML Message
1274 containing no ebXML Payload and the following elements in the ebXML Header:

- 1275 • A **MessageHeader** element
- 1276 • A **TraceHeaderList** element
- 1277 • A **Signature** element

1278 The **TraceHeaderList** and the **Signature** elements MAY be omitted (see sections 8.5 and
1279 8.13.7).

1280 The **MessageHeader** element MUST contain the following:

- 1281 • a **From** element that identifies the party creating the MSH Ping message
- 1282 • a **To** element that identifies the party that is being sent the MSH Ping message
- 1283 • a **Service** element that contains: **uri:www.ebxml.org/messageService/**
- 1284 • an **Action** element that contains **Ping**

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

1286 9.2.2 Message Service Handler Pong Message

1287 Once the **To** party receives the MSH Ping message, they MAY generate a Message Service
1288 Handler Pong (MSH Pong) message consisting of an ebXML Message containing no ebXML
1289 Payload and the following elements in the ebXML Header:

- 1290 • a **MessageHeader** element
- 1291 • a **TraceHeaderList** element
- 1292 • an **Acknowledgment** element
- 1293 • a **Signature** element

1294 The **TraceHeaderList**, **Acknowledgment** and **Signature** elements MAY be omitted (see
1295 sections 8.5, 8.13.6 and 8.13.7).

1296 The **MessageHeader** element MUST contain the following:

- 1297 • a **From** element that identifies the creator of the MSH Pong message
- 1298 • a **To** element that identifies a Party that generated the MSH Ping message
- 1299 • a **Service** element that contains the value: **uri:www.ebxml.org/messageService/**
- 1300 • an **Action** element that contains the value **Pong**
- 1301 • a **RefToMessageId** that identifies the MSH Ping message.

1302 The message is then sent to the **To** party.

1303 9.2.3 Security Considerations

1304 Parties who receive a MSH Ping message SHOULD always respond to the message. However,
1305 there is a risk that some parties might use the MSH Ping message to determine the existence of
1306 a Message Service Handler as part of a security attack on that MSH. Therefore, recipients of a
1307 MSH Ping MAY ignore the message if they consider that the sender of the message received is
1308 unauthorized or part of some attack. The decision process that results in this course of action is
1309 implementation dependent.

1310 10 Reliable Messaging

1311 Reliable Messaging defines an interoperable protocol such that the two Message Service
1312 Handlers (MSH) can “reliably” exchange messages that are sent using “reliable messaging”
1313 semantics, resulting in the *To Party* receiving the message once and only once.

1314 Reliability is achieved by a receiving MSH responding to a message with an *Acknowledgment*
1315 *Message*.

1316 10.1.1 Persistent Storage and System Failure

1317 A MSH that supports Reliable Messaging MUST keep messages that are sent or received reliably
1318 in *persistent storage*. In this context *persistent storage* is a method of storing data that does not
1319 lose information after a system failure or interruption.

1320 This specification recognizes that different degrees of resilience may be realized depending on
1321 the technology that is used to persist the data. However, as a minimum, persistent storage that
1322 has the resilience characteristics of a hard disk (or equivalent) SHOULD be used. It is strongly
1323 RECOMMENDED though that implementers of this specification use technology that is resilient to
1324 the failure of any single hardware or software component.

1325 After a system interruption or failure, an MSH MUST ensure that messages in persistent storage
1326 are processed in the same way as if the system failure or interruption had not occurred. How this
1327 is done is an implementation decision.

1328 In order to support the filtering of duplicate messages, a receiving MSH SHOULD save the
1329 **MessageId** in *persistent storage*. It is also RECOMMENDED that the following be kept in
1330 *Persistent Storage*:

- 1331 • the complete message, at least until the information in the message has been passed to
1332 the application or other process that needs to process it
- 1333 • the time the message was received, so that the information can be used to generate the
1334 response to a Message Status Request (see section 9.1)

1335 10.1.2 Methods of Implementing Reliable Messaging

1336 Support for Reliable Messaging MAY be implemented in one of the following two ways:

- 1337 • using the ebXML Reliable Messaging protocol, or
- 1338 • using ebXML Header and Message structures together with commercial software
1339 products that are designed to provide reliable delivery of messages using alternative
1340 protocols.

1341 10.2 Reliable Messaging Parameters

1342 This section describes the parameters required to control reliable messaging. This parameter
1343 information can be specified in the *CPA* or in the **MessageHeader** (section 8.4.2).

1344 10.2.1 Delivery Semantics

1345 The **deliverySemantics** value MUST be used by the *From Party* MSH to determine whether the
1346 Message must be sent reliably. Valid Values are:

- 1347 • **OnceAndOnlyOnce**. The message must be sent using a **reliableMessagingMethod**
1348 that will result in the application or other process at the *To Party* receiving the message
1349 once and only once
- 1350 • **BestEffort** The reliable delivery semantics are not used. In this case, the value of
1351 **reliableMessagingMethod** is ignored.

1352 The value for **deliverySemantics** is specified in the *CPA* or in **MessageHeader** (section 8.4.2).
1353 If no value is specified the default value is **BestEffort**.

1354 If **deliverySemantics** is set to **OnceAndOnlyOnce**, the *From Party* MSH and the *To Party* MSH
1355 must adopt the Reliable Messaging behavior (see section 10) that describes how messages are
1356 resent in the case of failure and duplicates are ignored.

1357 If **deliverySemantics** is set to **BestEffort**, a MSH that received a message that it is unable to
1358 deliver MUST NOT take any action to recover or otherwise notify anyone of the problem. The
1359 MSH that sent the message must not attempt to recover from any failure. This means that
1360 duplicate messages might be delivered to an application and persistent storage of messages is
1361 not required.

1362 If the *To Party* is unable to support the type of delivery semantics requested, the *To Party*
1363 SHOULD report the error to the *From Party* using an **ErrorCode** of **NotSupported** and a
1364 **Severity** of **Error**.

1365 10.2.2 MSHTimeAccuracy

1366 The **mshTimeAccuracy** parameter indicates the minimum accuracy a Receiving MSH keeps the
1367 clocks it uses when checking, for example, **TimeToLive**. Its value is in the format "mm:ss" which
1368 indicates the accuracy in minutes and seconds.

1369 10.2.3 Time To Live

1370 The **TimeToLive** value indicates the time by which a message should be delivered to and
1371 processed by the *To Party*. It must conform to an XML Schema `timeInstant`.

1372 In this context, the **TimeToLive** has expired if the time of the internal clock of the receiving MSH
1373 is greater than the value of **TimeToLive** for the message.

1374 When setting a value for **TimeToLive** it is RECOMMENDED that the *From Party*'s MSH takes
1375 into account the accuracy of its own internal clocks as well as the MSH **TimeAccuracy**
1376 parameter for the receiving MSH indicating the accuracy to which a MSH will keep its internal
1377 clocks. How a MSH ensures that its internal clocks are kept sufficiently accurate is an
1378 implementation decision.

1379 If the *To Party*'s MSH receives a message where **TimeToLive** has expired, it SHALL send a
1380 message to the *From Party* MSH, reporting that the **TimeToLive** of the message has expired.
1381 This message SHALL be comprised of an **ErrorList** containing an error that has the **errorCode**
1382 attribute set to **TimeToLiveExpired**, and the **severity** attribute set to **Error**.

1383 10.2.4 reliableMessagingMethod

1384 The **reliableMessagingMethod** attribute SHALL have one of the following values:

- 1385 • **ebXML**
- 1386 • **Transport**

1387 The default implied value for this attribute is **ebXML** and is case sensitive. Refer to section 8.6.5
1388 for discussion of the use of this attribute.

1389 10.2.5 AckRequested

1390 The **AckRequested** value is used by the sending MSH to request that the receiving MSH returns
1391 an *acknowledgment message* with an **Acknowledgment** element with a **type** of
1392 **Acknowledgment**.

1393 Valid values for **AckRequested** are:

- 1394 • **Unsigned** - requests that an unsigned Acknowledgement is requested
- 1395 • **Signed** - requests that a signed Acknowledgement is requested, or
- 1396 • **None** - indicates that no Acknowledgement is requested.

1397 The default value is **None**.

1398 **10.2.6 Timeout Parameter**

1399 The *timeout* parameter is an integer value that specifies the time expressed as a [XMLSchema]
 1400 timeDuration, that the Sending MSH MUST wait for an Acknowledgement Message before first
 1401 resending a message to the Receiving MSH.

1402 **10.2.7 Retries**

1403 The *retries* value is an integer value that specifies the maximum number of times a Sending
 1404 MSH SHOULD attempt to redeliver an unacknowledged *message* using the same
 1405 Communications Protocol.

1406 **10.2.8 RetryInterval**

1407 The *retryInterval* value is a time value, expressed as a duration in accordance with the
 1408 [XMLSchema] timeDuration data type. This value specifies the minimum time the Sending MSH
 1409 MUST wait between retries, if an *Acknowledgment Message* is not received.

1410 **10.2.9 PersistDuration**

1411 The *persistDuration* value is the minimum length of time, expressed as a [XMLSchema]
 1412 timeDuration, that data from a reliably sent *Message*, is kept in *Persistent Storage* by a receiving
 1413 MSH.

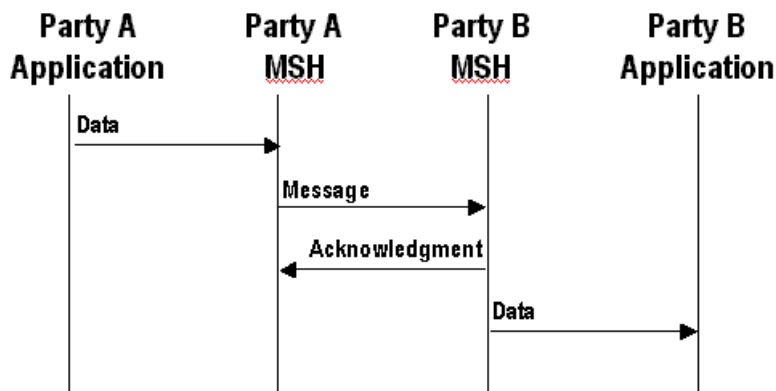
1414 If the *persistDuration* has passed since the message was first sent, a Sending MSH SHOULD
 1415 NOT resend a message with the same *MessageId*.

1416 If a message cannot be sent successfully before *persistDuration* has passed, then the Sending
 1417 MSH should report a delivery failure (see section 10.4).

1418 **10.3 ebXML Reliable Messaging Protocol**

1419 The ebXML Reliable Messaging Protocol described in this section MUST be followed if the
 1420 *deliverySemantics* parameter/element is set to *OnceAndOnlyOnce* and the
 1421 *reliableMessagingMethod* parameter/element is set to *ebXML* (the default).

1422 The ebXML Reliable Messaging Protocol is illustrated by the figure below.



1423

1424 **Figure 10-1 Indicating that a message has been received**

1425 The receipt of the *acknowledgment message* indicates that the message being acknowledged
 1426 has been successfully received and either processed or persisted by the receiving MSH.

1427 An *acknowledgment message* MUST contain a *MessageData* element with a *RefToMessageId*
 1428 that contains the same value as the *MessageId* element in the *message being acknowledged*.

1429 **10.3.1.1 Sending Message Behavior**

1430 If a MSH is given data by an application that needs to be sent reliably (i.e. the
1431 **deliverySemantics** is set to **OnceAndOnlyOnce**), then the MSH MUST do the following:

- 1432 1. Create a message from components received from the application that includes a
1433 **TraceHeader** element that identifies the sender and the receiver URIs.
- 1434 2. Save the message in *persistent storage* (see section 10.1.1)
- 1435 3. Send the message to the *Receiver* MSH
- 1436 4. Wait for the *Receiver* MSH to return an *acknowledgment message* and, if it does not, then
1437 take the appropriate action as described in section 10.3.1.4

1438 **10.3.1.2 Receiving Message Behavior**

1439 If the **deliverySemantics** for the received message is set to **OnceAndOnlyOnce** then do the
1440 following:

- 1441 1) If the message is just an acknowledgement (i.e. the **Service** element is set to
1442 <http://www.ebxml.org/namespaces/messageService/MessageAcknowledgment> and **Action**
1443 is set to **Acknowledgment**), then:
 - 1444 a) Look for a message in *persistent storage* that has a **MessageId** that is the same as the
1445 value of **RefToMessageId** on the received Message
 - 1446 b) If a message is found in *persistent storage* then mark the persisted message as delivered
- 1447 2) Otherwise, if the message is not just an acknowledgement, then check to see if the
1448 message is a duplicate (e.g. there is a **MessageId** held in *persistent storage* that was
1449 received earlier that contains the same value for the **MessageId**)
 - 1450 c) If the message is not a duplicate then do the following:
 - 1451 i) Save the **MessageId** of the received message in *persistent storage*. As an
1452 implementation decision, the whole message MAY be stored if there are other
1453 reasons for doing so.
 - 1454 ii) If the received message contains a **RefToMessageId** element then do the following:
 - 1455 (1) Look for a message in *persistent storage* that has a **MessageId** that is the same
1456 as the value of **RefToMessageId** on the received Message
 - 1457 (2) If a message is found in *persistent storage* then mark the persisted message as
1458 delivered
 - 1459 iii) Generate an *Acknowledgment Message* in response (see section 10.3.1.3).
 - 1460 d) If the message is a duplicate, then do the following:
 - 1461 i) Look in persistent storage for the first response to the received message and resend
1462 it (i.e. it contains a **RefToMessageId** that matches the **MessageId** of the received
1463 message)
 - 1464 ii) If a message was found in *persistent storage* then resend the persisted message
1465 back to the MSH that sent the received message,
 - 1466 iii) If no message was found in *persistent storage*, then:
 - 1467 (1) if **syncReply** is set to **True** and if the CPA indicates an application response is
1468 included, ignore the received message (i.e. no message was generated in
1469 response to the message, or the processing of the earlier message is not yet
1470 complete)

1471 (2) if **syncReply** is set to **False** then generate an *Acknowledgement Message* (see
 1472 section 10.3.1.3).

1473 **10.3.1.3 Generating an Acknowledgement Message**

1474 An *Acknowledgement Message* MUST be generated whenever a message is received with:

- 1475 • **deliverySemantics** set to **OnceAndOnlyOnce** and
- 1476 • **reliableMessagingMethod** set to **ebXML** (the default).

1477 As a minimum, it MUST contain a **MessageData** element with a **RefToMessageld** that contains
 1478 the same value as the **MessageId** element in the *message being acknowledged*.

1479 If **ackRequested** in the **TraceHeader** of the received message is set to **Signed** or **Unsigned**
 1480 then the acknowledgement message MUST also contain an **Acknowledgement** element.

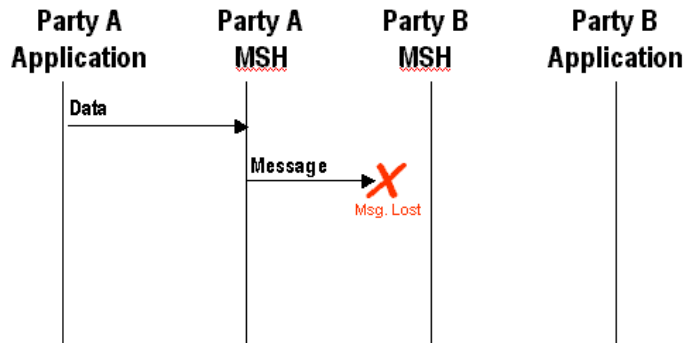
1481 Depending on the value of the **syncReply** parameter, the *Acknowledgement Message* can also
 1482 be sent at the same time as the response to the received message. In this case, the values for
 1483 the **MessageHeader** elements of the *Acknowledgement Message* are set by the designer of the
 1484 Service.

1485 If an **Acknowledgment** element is being sent on its own, then the value of the **MessageHeader**
 1486 elements MUST be set as follows:

- 1487 1) The **Service** element MUST be set to: **uri:www.ebxml.org/messageService/**
- 1488 2) The **Action** element MUST be set to **Acknowledgment**.
- 1489 3) The **From** element MAY be populated with the **TO** element extracted from the message
 1490 received, or it MAY be set to the **ReceiverURI** from the last **TraceHeader** in the *message*
 1491 that has just been received
- 1492 4) The **To** element MAY be populated with the **FROM** element extracted from the message
 1493 received, or it MAY be set to the **SenderURI** from the last **TraceHeader** in the *message* that
 1494 has just been received
- 1495 5) The **RefToMessageld** element MUST be set to the **MessageId** of the *message* that has just
 1496 been received

1497 **10.3.1.4 Resending Lost Messages and Duplicate Filtering**

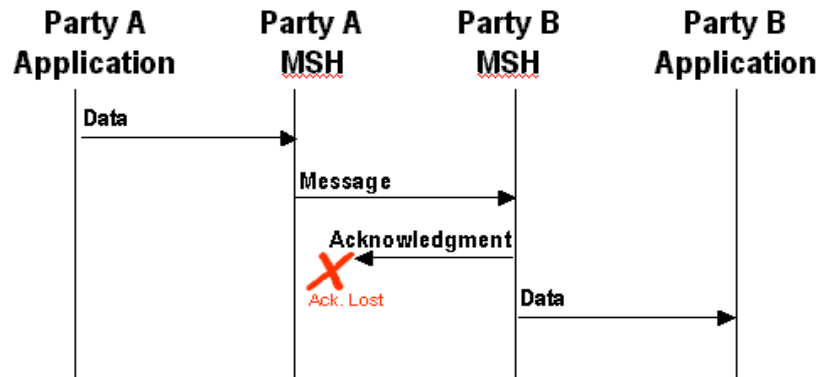
1498 This section describes the behavior that is required by the sender and receiver of a message in
 1499 order to handle when messages are lost. A message is "lost" when a sending MSH does not
 1500 receive a response to a message. For example, it is possible that a *message* was lost, for
 1501 example:



1502

1503 **Figure 10-2 Undelivered Message**

1504 It is also possible that the *Acknowledgment Message* was lost, for example:



1505

1506 **Figure 10-3 Lost Acknowledgment Message**

1507 The rules that apply are as follows:

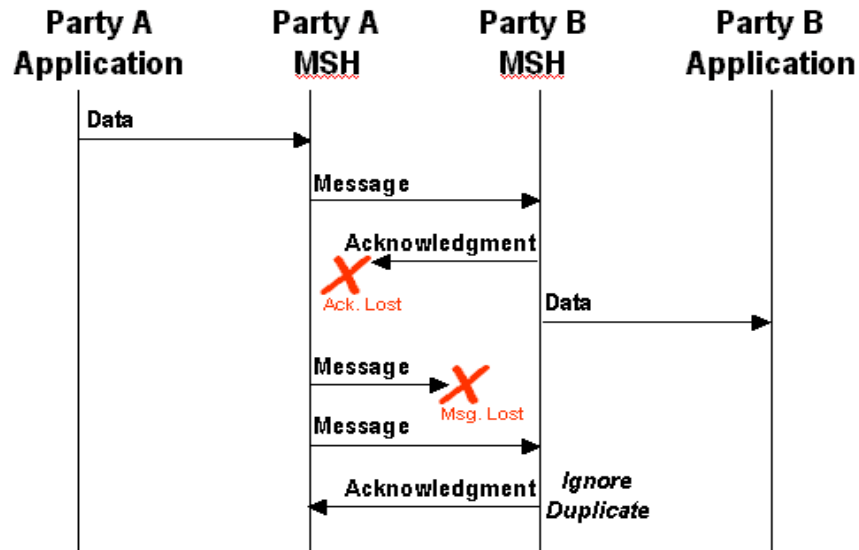
- 1508 1) The Sending MSH MUST resend the original message if an *Acknowledgment Message* has
 1509 not been received from the Receiving MSH and either of the following are true:
- 1510 a) The message has not yet been resent and at least the time specified in the **timeout**
 1511 parameter has passed since the first message was sent, or
 - 1512 b) The message has been resent, and the following are both true:
 - 1513 i) At least the time specified in the **retryInterval** has passed since the last time the
 1514 message was resent, and
 - 1515 ii) The message has been resent less than the number of times specified in the **retries**
 1516 Parameter
- 1517 2) If the Sending MSH does not receive an *Acknowledgment Message* after the maximum
 1518 number of retries, the Sending MSH SHOULD notify the application and/or system
 1519 administrator function of the failure to receive an acknowledgement.
- 1520 3) If the Sending MSH detects an unrecoverable communications protocol error at the transport
 1521 protocol level, the Sending MSH SHOULD resend the message.

1522 **10.3.1.5 Duplicate Message Handling**

1523 In this context:

- 1524 • an *identical message* is a message that contains, apart from perhaps an additional
 1525 **TraceHeader** element, the same *ebXML Header* and *ebXML Payload* as the earlier
 1526 message that was sent.
- 1527 • a *duplicate message* is a message that contains the same **MessageId** as an earlier
 1528 message that was received.
- 1529 • the first message is the message with the earliest **Timestamp** in the **MessageData**
 1530 element that has the same **RefToMessageId** as the duplicate message.

1531 Note: the Communication Protocol Envelope MAY be different. This means the same message MAY be
 1532 sent using different communication protocols and the reliable messaging behavior described in this section
 1533 will still apply. The ability to use alternative communication protocols may be specified in the CPA and is
 1534 an OPTIONAL implementation specific feature.



1535

1536 **Figure 10-4 Resending Unacknowledged Messages**

1537 The diagram above shows the behavior that MUST be followed by the sending and receiving
 1538 MSH that are sent with **deliverySemantics** of **OnceAndOnlyOnce**. Specifically:

- 1539 1) The sender of the *message* (e.g. Party A) MUST resend the *identical message* if no
 1540 *Acknowledgment Message* is received
- 1541 2) When the recipient (Party B) of the *message* receives a *duplicate message*, it MUST resend
 1542 to the sender (Party A) a message identical to the *first message* that was sent to the sender
 1543 Party A)
- 1544 3) The recipient of the *message* (Party B) MUST NOT forward the message a second time to
 1545 the application/process.

1546 **10.4 Failed Message Delivery**

1547 If a message sent with **deliverySemantics** set to **OnceAndOnlyOnce** cannot be delivered, the
 1548 MSH or process SHOULD send a delivery failure notification to the **From Party**. The delivery
 1549 failure notification message contains:

- 1550
- 1551 • a **From Party** that identifies the party who detected the problem
 - 1552 • a **To Party** that identifies the **From Party** that created the message that could not be delivered
 - 1553 • a **Service** element and **Action** element set as described in 11.5
 - 1554 • an **Error** element with a severity of:
 - 1555 - **Error** if the party who detected the problem could not transmit the message (e.g. the
 1556 communications transport was not available)
 - 1557 - **Warning** if the message was transmitted, but an *acknowledgment message* was not
 1558 received. This means that the message probably was not delivered although there is
 1559 a small probability that it was.
 - 1560 • an **ErrorCode** of **DeliveryFailure**

1561 It is possible that an error message with an **Error** element with an **ErrorCode** set to
 1562 **DeliveryFailure** cannot be delivered successfully for some reason. If this occurs, then the From
 1563 Party that is the ultimate destination for the error message SHOULD be informed of the problem
 1564 by other means. How this is done is outside the scope of this specification.

1565 11 Error Reporting and Handling

1566 This section describes how one ebXML Message Service Handler (MSH) reports errors it detects
1567 in an ebXML Message to another MSH. The *ebXML Message Service* error reporting and
1568 handling is to be considered as being a layer of processing above the SOAP Processor layer.
1569 This means that the ebXML MSH is essentially an application-level handler of a *SOAP message*
1570 from the perspective of the SOAP Processor. The SOAP Processor MAY generate SOAP Fault
1571 messages if it is unable to process the message. A Sending MSH MUST be prepared to accept
1572 and process these SOAP Faults.

1573 It is possible for the ebXML MSH software to cause a SOAP Fault to be generated and returned
1574 to the sender of a *SOAP message*. In this event, the returned message MUST conform to the
1575 [SOAP] specification processing guidelines for SOAP Faults.

1576 An ebXML *SOAP message* that reports an error that has a **highestSeverity** of **Warning** SHALL
1577 NOT be reported or returned as a SOAP Fault.

1578 11.1 Definitions

1579 For clarity two phrases are defined that are used in this section:

- 1580 • *message in error*. A *message* that contains or causes an error of some kind
- 1581 • *message reporting the error*. A *message* that contains an ebXML **ErrorList** element that
1582 describes the error(s) found in a *message in error*.

1583 11.2 Types of Errors

1584 One MSH needs to report to another MSH errors in a *message in error*. For example, errors
1585 associated with:

- 1586 • ebXML namespace qualified content of the *SOAP message* document (see section 8),
- 1587 • reliable messaging failures (see section 10), or
- 1588 • security (see section 12).

1589 Unless specified to the contrary, all references to "an error" in the remainder of this specification
1590 imply any or all of the types of errors listed above.

1591 Errors associated with Data Communication protocols are detected and reported using the
1592 standard mechanisms supported by that data communication protocol and are do not use the
1593 error reporting mechanism described here.

1594 11.3 When to generate Error Messages

1595 When an MSH detects an error in a message it is strongly RECOMMENDED that the error is
1596 reported to the MSH that sent the message that had an error if:

- 1597 • the Error Reporting Location (see section 11.4) to which the *message reporting the error*
1598 should be sent can be determined, and
- 1599 • the *message in error* does not have an **ErrorList** element with **highestSeverity** set to
1600 **Error**.

1601 If the Error Reporting Location cannot be found or the *message in error* has an **ErrorList** element
1602 with **highestSeverity** set to **Error**, it is RECOMMENDED that:

- 1603 • the error is logged,
- 1604 • the problem is resolved by other means, and
- 1605 • no further action is taken.

1606 11.3.1 Security Considerations

1607 Parties that receive a Message containing an error in the header SHOULD always respond to the
1608 message. However they MAY ignore the message and not respond if they consider that the
1609 message received is unauthorized or is part of some security attack. The decision process that
1610 results in this course of action is implementation dependent.

1611 11.4 Identifying the Error Reporting Location

1612 The Error Reporting Location is a URI that is specified by the sender of the *message in error* that
1613 indicates where to send a *message reporting the error*.

1614 The **ErrorURI** implied by the CPA, identified by the **CPAId** on the message, SHOULD be used. If
1615 no **ErrorURI** is implied by the CPA, the **SenderURI** MUST be used.

1616 Even if the *message in error* cannot be successfully analyzed or parsed, MSH implementers
1617 SHOULD try to determine the Error Reporting Location by other means. How this is done is an
1618 implementation decision.

1619 11.5 Service and Action Element Values

1620 An **ErrorList** element can be included in an **ebXMLHeader** that is part of a *message* that is being
1621 sent as a result of processing of an earlier message. In this case, the values for the **Service** and
1622 **Action** elements are set by the designer of the Service.

1623 An **ErrorList** element can also be included in an **ebXMLHeader** that is not being sent as a result
1624 of the processing of an earlier message. In this case, the values of the **Service** and **Action**
1625 elements MUST be set as follows if the **highestSeverity** is set to **Error**. If the **highestSeverity**
1626 is set to **Warning**, the **Service** and **Action** elements MUST NOT be used.

- 1627 • The **Service** element MUST be set to: **uri:www.ebxml.org/messageService/**
- 1628 • The **Action** element MUST be set to **MessageError**.

1629 **12 Security**

1630 The *ebXML Message Service*, by its very nature, presents certain security risks. A Message
1631 Service may be at risk by means of:

- 1632 • Unauthorized access
- 1633 • Data integrity and/or confidentiality attacks (e.g. through man-in-the-middle attacks)
- 1634 • Denial-of-Service, spoofing, bombing attacks

1635 Each security risk is described in detail in the ebXML Technical Architecture Security
1636 Specification [EBXMLSEC].

1637 Each of these security risks MAY be addressed in whole, or in part, by the application of one, or a
1638 combination, of the countermeasures described in this section. This specification describes a set
1639 of profiles, or combinations of selected countermeasures, that have been selected to address key
1640 risks based upon commonly available technologies. Each of the specified profiles includes a
1641 description of the risks that are not addressed.

1642 Application of countermeasures SHOULD be balanced against an assessment of the inherent
1643 risks and the value of the asset(s) that might be placed at risk.

1644 **12.1 Security and Management**

1645 No technology, regardless of how advanced it might be, is an adequate substitute to the effective
1646 application of security management policies and practices.

1647 It is strongly RECOMMENDED that the site manager of an *ebXML Message Service* apply due
1648 diligence to the support and maintenance of its; security mechanism, site (or physical) security
1649 procedures, cryptographic protocols, update implementations and apply fixes as appropriate.
1650 (See <http://www.cert.org/> and <http://ciac.llnl.gov/>)

1651 **12.2 Collaboration Protocol Agreement**

1652 The configuration of Security for MSHs may be specified in the CPA. Three areas of the CPA
1653 have security definitions as follows:

- 1654 • The Document Exchange section addresses security to be applied to the payload of the
1655 message. The MSH is not responsible for any security specified at this level but may
1656 offer these services to the message sender.
- 1657 • The Message section addresses security applied to the entire ebXML Document, which
1658 includes the header and the payload.
- 1659 • The Transport section addresses the Transport level. The MSH is not responsible for
1660 any security specified at this level.

1661 **12.3 Countermeasure Technologies**

1662 **12.3.1 Persistent Digital Signature**

1663 If signatures are being used to digitally sign an ebXML Message then XML Signature [DSIG]
1664 MUST be used to bind the ebXML Header Document to the ebXML Payload or data elsewhere on
1665 the web that relates to the message. It is also strongly RECOMMENDED that XML Signature is
1666 used to digitally sign the Payload on its own.

1667 The only available technology that can be applied to the purpose of digitally signing an ebXML
1668 Message (both the ebXML Header and its associated payload objects) is provided by technology
1669 that conforms to the W3C/IETF joint XML Signature specification [XMLDSIG]. An XML Signature
1670 conforming to this specification can selectively sign portions of an XML document(s), permitting

1671 the documents to be augmented (new element content added) while preserving the validity of the
1672 signature(s).

1673 An ebXML Message that requires a digital signature SHALL be signed following the process
1674 defined in this section of the specification and SHALL be in full compliance with [XMLDSIG].

1675 12.3.1.1 Signature Generation

1676 1) Create a **SignedInfo** element with SignatureMethod, CanonicalizationMethod, and
1677 Reference(s) elements for the ebXML Header document and any required payload objects,
1678 as prescribed by [XMLDSIG].

1679 2) Canonicalize and then calculate the SignatureValue over **SignedInfo** based on algorithms
1680 specified in SignedInfo as specified in [XMLDSIG].

1681 3) Construct the Signature element that includes the **SignedInfo**, **KeyInfo** (RECOMMENDED),
1682 and **SignatureValue** elements as specified in [XMLDSIG].

1683 4) Include the namespace qualified **Signature** element in the ebXML Header document just
1684 signed, following the **TraceHeaderList** element.

1685 The **ds:SignedInfo** element SHALL be composed of zero or one **ds:CanonicalizationMethod**
1686 element, the **ds:SignatureMethod** and one or more **ds:Reference** elements.

1687 The **ds:CanonicalizationMethod** element is defined as OPTIONAL in [XMLDSIG], meaning that
1688 the element need not appear in an instance of a **ds:SignedInfo** element. The default
1689 canonicalization method that is applied to the data to be signed is [XMLC14N] in the absence of a
1690 **ds:Canonicalization** element that specifies otherwise. This default SHALL also serve as the
1691 default canonicalization method for the *ebXML Message Service*.

1692 The **ds:SignatureMethod** element SHALL be present and SHALL have an Algorithm attribute.
1693 The RECOMMENDED value for the Algorithm attribute is:

1694 <http://www.w3.org/2000/02/xmlsig#sha1>

1695 This RECOMMENDED value SHALL be supported by all compliant *ebXML Message Service*
1696 software implementations.

1697 The **ds:Reference** element for the ebXML Header document SHALL have a URI attribute value
1698 of "" to provide for the signature to be applied to the document that contains the **ds:Signature**
1699 element (the ebXML Header document). The **ds:Reference** element for the ebXML Header
1700 document MAY include a **Type** attribute that has a value
1701 "http://www.w3.org/2000/02/xmlsig#Object" in accordance with [XMLDSIG]. This attribute is
1702 purely informative. It MAY be omitted. Implementations of the ebXML MSH SHALL be prepared
1703 to handle either case. The **ds:Reference** element MAY include the optional **id** attribute.

1704 The **ds:Reference** element for the ebXML Header document SHALL include a child
1705 **ds:Transform** element that excludes the containing **ds:Signature** element and all its
1706 descendants as well as the **TraceHeaderList** element and all its descendants as these elements
1707 are subject to change. The **ds:Transform** element SHALL include a child **ds:XPath** element that
1708 has a value of:
1709 [/descendant-or-self::node\(\)\[not\(ancestor-or-self::ds:Signature\[@id='S1'\]\)\]](#) and not (ancestor-or-
1710 self::TraceHeaderList])

1711 Each payload object that requires signing SHALL be represented by a **ds:Reference** element
1712 that SHALL have a **URI** attribute that resolves to that payload object. This MAY be either the
1713 Content-Id URI of the payload object enveloped in the MIME ebXML Payload Container, or a
1714 URI that matches the Content-Location header of the payload object enveloped in the ebXML
1715 Payload Container, or a URI that resolves to an external payload object that is external to the
1716 ebXML Payload Container. It is strongly RECOMMENDED that the URI attribute value match the
1717 xlink:href URI value of the corresponding **Manifest/Reference** element for that payload object.
1718 However, this is NOT REQUIRED.

1719 Example of digitally signed ebXML SOAP message:

```

1720
1721 <?xml version="1.0" encoding="utf-8"?>
1722 <SOAP-ENV:Envelope
1723   xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
1724   xmlns:eb="http://www.ebxml.org/namespaces/messageHeader"
1725   xmlns:xlink="http://www.w3.org/1999/xlink">
1726   <SOAP-ENV:Header>
1727     <eb:MessageHeader id="..." eb:version="98.0">
1728       ...
1729     </eb:MessageHeader>
1730     <eb:TraceHeaderList id="..." eb:version="98.0">
1731       <eb:TraceHeader>
1732         ...
1733       </eb:TraceHeader>
1734     </eb:TraceHeaderList>
1735     <ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmlds#">
1736       <ds:SignedInfo>
1737         <ds:CanonicalizationMethod Algorithm="http://www.w3.org/TR/2000/WD-xml-c14n-20001011"/>
1738         <ds:SignatureMethod Algorithm="http://www.w3.org/2000/09/xmlds#dsa-sha1"/>
1739         <ds:Reference URI="">
1740           <ds:Transforms>
1741             <ds:Transform>
1742               <XPath>/descendant-or-self::node()[not(ancestor-or-self::ds:Signature[@id='S1'])
1743 and not(ancestor-or-self::TraceHeaderList)]</XPath>
1744             </ds:Transform>
1745           </ds:Transforms>
1746           <ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmlds#sha1"/>
1747           <ds:DigestValue>...</ds:DigestValue>
1748         </ds:Reference>
1749         <ds:Reference URI="cid://blahblahblah/">
1750           <ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmlds#sha1"/>
1751           <ds:DigestValue>...</ds:DigestValue>
1752         </ds:Reference>
1753       </ds:SignedInfo>
1754       <ds:SignatureValue>...</ds:SignatureValue>
1755       <ds:KeyInfo>...</ds:KeyInfo>
1756     </ds:Signature>
1757   </SOAP-ENV:Header>
1758   <SOAP-ENV:Body>
1759     <eb:Manifest id="Mani01" eb:version="98.0">
1760       <eb:Reference xlink:href="cid://blahblahblah"
1761         xlink:role="http://ebxml.org/gci/invoice">
1762         <eb:Schema eb:version="98.0" eb:location="http://ebxml.org/gci/busdocs/invoice.dtd"/>
1763       </eb:Reference>
1764     </eb:Manifest>
1765   </SOAP-ENV:Body>
1766 </SOAP-ENV:Envelope>

```

1767

1768 12.3.2 Persistent Signed Receipt

1769 An ebXML Message that has been digitally signed MAY be acknowledged with a DeliveryReceipt
 1770 acknowledgment message that itself is digitally signed in the manner described in the previous
 1771 section. The acknowledgment message MUST contain the set of **ds:DigestValue** elements
 1772 contained in the **ds:Signature** element of the original message within the **Acknowledgment**
 1773 element.

1774 12.3.3 Non-persistent Authentication

1775 Non-persistent authentication is provided by the communications channel used to transport the
 1776 ebXML Message. This authentication MAY be either in one direction—from the session initiator to
 1777 the receiver—or bi-directional. The specific method will be determined by the communications
 1778 protocol used. For instance, the use of a secure network protocol, such as [RFC2246] or [IPSEC]
 1779 provides the sender of an ebXML Message to authenticate the destination for the TCP/IP
 1780 environment.

1781 12.3.4 Non-persistent Integrity

1782 Use of a secure network protocol such as [RFC2246] or [IPSEC] MAY be configured so as to
1783 provide for integrity check CRCs of the packets transmitted via the network connection.

1784 12.3.5 Persistent Confidentiality

1785 XML Encryption is a W3C/IETF joint activity that is actively engaged in the drafting of a
1786 specification for the selective encryption of an XML document(s). It is anticipated that this
1787 specification will be completed within the next year. The ebXML Transport, Routing and
1788 Packaging team has identified this technology as the only viable means of providing persistent,
1789 selective confidentiality of elements within an ebXML Message including the ebXML Header
1790 document.

1791 Confidentiality for ebXML Payloads MAY be provided by functionality possessed by a MSH.
1792 However, this specification states that it is not the responsibility of the MSH to provide security for
1793 the ebXML Payloads. Payload confidentiality MAY be provided by using XML Encryption (when
1794 available) or some other cryptographic process, such as [S/MIME], [S/MIMEV3], or [PGP/MIME],
1795 that is bilaterally agreed upon by the parties involved. Since XML Encryption is not currently
1796 available, it is RECOMMENDED that [S/MIME] encryption methods be used for ebXML Payloads.
1797 The XML Encryption standard SHALL be the default encryption method when XML Encryption
1798 has achieved W3C Recommendation status.

1799 12.3.6 Non-persistent Confidentiality

1800 Use of a secure network protocol such as [RFC2246] or [IPSEC] provides transient confidentiality
1801 of a message as it is transferred between two ebXML MSH nodes.

1802 12.3.7 Persistent Authorization

1803 The OASIS Security Services TC is actively engaged in the definition of a specification that
1804 provides for the exchange of security credentials, including NameAssertion and Entitlements that
1805 is based on [S2ML]. Use of technology that is based on this anticipated specification MAY be
1806 used to provide persistent authorization for an ebXML Message once it becomes available.
1807 ebXML has a formal liaison to this TC. There are also many ebXML member organizations and
1808 contributors that are active members of the OASIS Security Services TC such as Sun, IBM,
1809 CommerceOne, Cisco and others that are endeavoring to ensure that the specification meets the
1810 requirements of providing persistent authorization capabilities for the *ebXML Message Service*.

1811 12.3.8 Non-persistent Authorization

1812 Use of a secure network protocol such as [RFC2246] or [IPSEC] MAY be configured to provide
1813 for bilateral authentication of certificates prior to establishing a session. This provides for the
1814 ability for an ebXML MSH to authenticate the source of a connection that can be used to
1815 recognize the source as an authorized source of ebXML Messages.

1816 12.3.9 Trusted Timestamp

1817 At the time of this specification, services that offer trusted timestamp capabilities are becoming
1818 available. Once these become more widely available, and a standard has been defined for their
1819 use and expression, these standards, technologies and services will be evaluated and considered
1820 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 authorizes sender based on communication channel credentials.
	Profile 3		✓				✓				sending MSH authenticates and both MSHs negotiate a 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)

1821

1822 **13 References**1823 **13.1 Normative References**

- 1824 [HTTP] IETF RFC 2068 - Hypertext Transfer Protocol -- HTTP/1.1, R. Fielding, J.
1825 Gettys, J. Mogul, H. Frystyk, T. Berners-Lee, January 1997
- 1826 [RFC822] Standard for the format of ARPA Internet text messages. D. Crocker. Aug-
1827 13-1982.
- 1828 [RFC2045] IETF RFC 2045. Multipurpose Internet Mail Extensions (MIME) Part One:
1829 Format of Internet Message Bodies, N Freed & N Borenstein, Published
1830 November 1996
- 1831 [RFC2046] Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types. N.
1832 Freed, N. Borenstein. November 1996.
- 1833 [RFC2246] RFC 2246 - Dierks, T. and C. Allen, "The TLS Protocol", January 1999.
- 1834 [RFC2387] The MIME Multipart/Related Content-type. E. Levinson. August 1998.
- 1835 [RFC2392] IETF RFC 2392. Content-ID and Message-ID Uniform Resource Locators. E.
1836 Levinson, Published August 1998
- 1837 [RFC2396] IETF RFC 2396. Uniform Resource Identifiers (URI): Generic Syntax. T
1838 Berners-Lee, Published August 1998
- 1839 [RFC2487] SMTP Service Extension for Secure SMTP over TLS. P. Hoffman.
1840 January 1999.
- 1841 [RFC2554] SMTP Service Extension for Authentication. J. Myers. March 1999.
- 1842 [RFC2616] RFC 2616 - Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L.,
1843 Leach, P. and T. Berners-Lee, "Hypertext Transfer Protocol, HTTP/1.1", ,
1844 June 1999.
- 1845 [RFC2617] RFC2617 - Franks, J., Hallam-Baker, P., Hostetler, J., Lawrence, S., Leach,
1846 P., Luotonen, A., Sink, E. and L. Stewart, "HTTP Authentication: Basic and
1847 Digest Access Authentication", June 1999.
- 1848 [RFC2817] RFC 2817 - Khare, R. and S. Lawrence, "Upgrading to TLS Within
1849 HTTP/1.1", May 2000.
- 1850 [RFC2818] RFC 2818 - Rescorla, E., "HTTP Over TLS", , May 2000.[SOAP] Simple
1851 Object Access Protocol
- 1852 [SMTP] IETF RFC 821, Simple Mail Transfer Protocol, J Postel, August 1982
- 1853 [SOAP] W3C-Draft-Simple Object Access Protocol (SOAP) v1.1, Don Box,
1854 DevelopMentor; David Ehnebuske, IBM; Gopal Kakivaya, Andrew Layman,
1855 Henrik Frystyk Nielsen, Satish Thatte, Microsoft; Noah Mendelsohn, Lotus
1856 Development Corp.; Dave Winer, UserLand Software, Inc.; W3C Note 08
1857 May 2000, <http://www.w3.org/TR/SOAP>
- 1858 [SOAPATTACH] SOAP Messages with Attachments, John J. Barton, Hewlett Packard Labs;
1859 Satish Thatte and Henrik Frystyk Nielsen, Microsoft, Published Oct 09 2000
1860 <http://www.w3.org/TR/SOAP-attachments>
- 1861 [SSL3] A. Frier, P. Karlton, and P. Kocher, "The SSL 3.0 Protocol", Netscape
1862 Communications Corp., Nov 18, 1996.
- 1863 [UTF-8] UTF-8 is an encoding that conforms to ISO/IEC 10646. See [XML] for usage
1864 conventions.

- 1865 [XLINK] W3C XML Linking Candidate Recommendation, <http://www.w3.org/TR/xlink/>
- 1866 [XML] W3C Recommendation: Extensible Markup Language (XML) 1.0 (Second
1867 Edition), October 2000, <http://www.w3.org/TR/2000/REC-xml-20001006>
- 1868 [XML Namespace] W3C Recommendation for Namespaces in XML, World Wide Web
1869 Consortium, 14 January 1999, <http://www.w3.org/TR/REC-xml-names>
- 1870 [XMLDSIG] Joint W3C/IETF XML-Signature Syntax and Processing specification,
1871 <http://www.w3.org/TR/2000/CR-xmlsig-core-20001031/>
- 1872 [XMLMedia] IETF RFC 3023, XML Media Types. M. Murata, S. St.Laurent, January 2001
- 1873 **13.2 Non-Normative References**
- 1874 [EBXMLTP] ebXML Collaboration Protocol Profile and Agreement specification, Version
1875 0.92, published 3 March, 2001
- 1876 [EBXMLTA] ebXML Technical Architecture, version 1.04 published 16 February, 2001
- 1877 [EBXMLTASEC] ebXML Technical Architecture Security Specification, version 0.3 published
1878 19 February, 2001
- 1879 [EBXMLRSS] ebXML Registry Services Specification, version 0.84
- 1880 [EBXMLMSREQ] ebXML Transport, Routing and Packaging: Overview and Requirements,
1881 Version 0.96, Published 25 May 2000
- 1882 [Glossary] ebXML Glossary, see ebXML Project Team Home Page
1883 [http://www.ebxml.org/project_teams/technical_coord/ebxml_ta_glossary95.xl](http://www.ebxml.org/project_teams/technical_coord/ebxml_ta_glossary95.xls)
1884 [s](http://www.ebxml.org/project_teams/technical_coord/ebxml_ta_glossary95.xls)
- 1885 [IPSEC] IETF RFC2402 IP Authentication Header. S. Kent, R. Atkinson. November
1886 1998. RFC2406 IP Encapsulating Security Payload (ESP). S. Kent, R.
1887 Atkinson. November 1998.
- 1888 [PGP/MIME] IETF RFC2015, "MIME Security with Pretty Good Privacy (PGP)", M. Elkins.
1889 October 1996.
- 1890 [S/MIME] IETF RFC2311, "S/MIME Version 2 Message Specification", S. Dusse, P.
1891 Hoffman, B. Ramsdell, L. Lundblade, L. Repka. March 1998.
- 1892 [S/MIMECH] IETF RFC 2312, "S/MIME Version 2 Certificate Handling", S. Dusse, P.
1893 Hoffman, B. Ramsdell, J. Weinstein. March 1998.
- 1894 [S/MIMEV3] IETF RFC 2633 S/MIME Version 3 Message Specification. B. Ramsdell, Ed..
1895 June 1999.
- 1896 [TLS] RFC2246, T. Dierks, C. Allen. January 1999.
- 1897 [XMLSchema] W3C XML Schema Candidate Recommendation,
1898 <http://www.w3.org/TR/xmlschema-0/>
1899 <http://www.w3.org/TR/xmlschema-1/>
1900 <http://www.w3.org/TR/xmlschema-2/>
- 1901 [XMTP] XMTP - Extensible Mail Transport Protocol
1902 <http://www.openhealth.org/documents/xmtp.htm>

1903 **14 Disclaimer**

1904 The views and specification expressed in this document are those of the authors and are not
1905 necessarily those of their employers. The authors and their employers specifically disclaim
1906 responsibility for any problems arising from correct or incorrect implementation or use of this
1907 design.

1908 **15 Contact Information**1909 **Team Leader**

1910 Name Rik Drummond
1911 Company Drummond Group, Inc.
1912 Street 5008 Bentwood Ct.
1913 City, State, Postal Code Fort Worth, Texas 76132
1914 Country USA
1915 Phone +1 (817) 294-7339
1916 EMail: rik@drummondgroup.com

1917

1918 **Vice Team Leader**

1919 Name Christopher Ferris
1920 Company Sun Microsystems
1921 Street One Network Drive
1922 City, State, Postal Code Burlington, MA 01803-0903
1923 Country USA
1924 Phone: +1 (781) 442-3063
1925 EMail: chris.ferris@sun.com

1926

1927 **Team Editor**

1928 Name David Burdett
1929 Company Commerce One
1930 Street 4400 Rosewood Drive
1931 City, State, Postal Code Pleasanton, CA 94588
1932 Country USA
1933 Phone: +1 (925) 520-4422
1934 EMail: david.burdett@commerceone.com

1935

1936 **Authors**

1937 Name Dick Brooks
1938 Company Group 8760
1939 Street 110 12th Street North, Suite F103
1940 City, State, Postal Code Birmingham, Alabama 35203
1941 Phone: +1 (205) 250-8053
1942 Email: dick@8760.com

1943

1944 Name David Burdett
1945 Company Commerce One
1946 Street 4400 Rosewood Drive
1947 City, State, Postal Code Pleasanton, CA 94588
1948 Country USA
1949 Phone: +1 (925) 520-4422
1950 EMail: david.burdett@commerceone.com

1951

1952 Name Christopher Ferris
1953 Company Sun Microsystems
1954 Street One Network Drive
1955 City, State, Postal Code Burlington, MA 01803-0903
1956 Country USA
1957 Phone: +1 (781) 442-3063
1958 EMail: chris.ferris@east.sun.com

1959

1960 Name John Ibbotson
1961 Company IBM UK Ltd

1962 Street Hursley Park
1963 City, State, Postal Code Winchester SO21 2JN
1964 Country United Kingdom
1965 Phone: +44 (1962) 815188
1966 Email: john_ibbotson@uk.ibm.com
1967
1968 Name Masayoshi Shimamura
1969 Company Fujitsu Limited
1970 Street Shinyokohama Nikko Bldg., 15-16, Shinyokohama 2-chome
1971 City, State, Postal Code Kohoku-ku, Yokohama 222-0033, Japan
1972 Phone: +81-45-476-4590
1973 EMail: shima@rp.open.cs.fujitsu.co.jp
1974
Document Editing Team
1975
1976 Name Ralph Berwanger
1977 Company bTrade.com
1978 Street 2324 Gateway Drive
1979 City, State, Postal Code Irving, TX 75063
1980 Country USA
1981 Phone: +1 (972) 580-3970
1982 EMail: rberwanger@btrade.com
1983
1984 Name Colleen Evans
1985 Company Progress/Sonic Software
1986 Street 14 Oak Park
1987 City, State, Postal Code Bedford, MA 01730
1988 Country USA
1989 Phone +1 (720) 480-3919
1990 Email cevans@progress.com
1991
1992 Name Ian Jones
1993 Company British Telecommunications
1994 Street Enterprise House, 84-85 Adam Street
1995 City, State, Postal Code Cardiff, CF24 2XF
1996 Country United Kingdom
1997 Phone: +44 29 2072 4063
1998 EMail: ian.c.jones@bt.com
1999
2000 Name Martha Warfelt
2001 Company DaimlerChrysler Corporation
2002 Street 800 Chrysler Drive
2003 City, State, Postal Code Auburn Hills, MI
2004 Country USA
2005 Phone: +1 (248) 944-5481
2006 EMail: maw2@daimlerchrysler.com
2007
2008 Name David Fischer
2009 Company Drummond Group, Inc
2010 Street 5008 Bentwood Ct
2011 City, State, Postal Code Fort Worth, TX 76132
2012 Phone +1 (817-294-7339
2013 EMail david@drummondgroup.com

2014 Appendix A ebXML SOAP Extension Elements Schema

2015 The following is the definition of the ebXML SOAP Header extension elements as a schema that
 2016 conforms to [XMLSchema]. This schema is a normative definition of the ebXML SOAP extension
 2017 elements. Section 8 details the normative specification for these elements.

2018 Note: if inconsistencies exist between the specification and this schema, the specification supersedes this
 2019 example schema.

```

2020
2021 <?xml version="1.0" encoding="UTF-8"?>
2022 <xsd:schema xmlns="http://www.ebxml.org/namespaces/messageHeader"
2023 targetNamespace="http://www.ebxml.org/namespaces/messageHeader"
2024 xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/" version="0.98b"
2025 xmlns:xlink="http://www.w3.org/1999/xlink" xmlns:xsd="http://www.w3.org/2000/10/XMLSchema">
2026   <xsd:import namespace="http://www.w3.org/1999/xlink"
2027     schemaLocation="http://www.w3.org/1999/xlink"/>
2028   <xsd:import namespace="http://schemas.xmlsoap.org/soap/envelope/"
2029     schemaLocation="http://schemas.xmlsoap.org/soap/envelope"/>
2030   <!-- MANIFEST -->
2031   <xsd:element name="Manifest">
2032     <xsd:complexType>
2033       <xsd:sequence>
2034         <xsd:element ref="Reference" maxOccurs="unbounded"/>
2035         <xsd:any namespace="##other" processContents="lax" minOccurs="0" maxOccurs="unbounded"/>
2036       </xsd:sequence>
2037       <xsd:attribute name="id" use="required" type="xsd:ID"/>
2038       <xsd:attribute name="version" use="fixed" type="xsd:string" value="98.0"/>
2039       <xsd:attribute ref="soap:mustUnderstand" use="required"/>
2040     </xsd:complexType>
2041   </xsd:element>
2042   <xsd:element name="Reference">
2043     <xsd:complexType>
2044       <xsd:sequence>
2045         <xsd:element ref="Schema" minOccurs="0" maxOccurs="unbounded"/>
2046         <xsd:element ref="Description" minOccurs="0" maxOccurs="unbounded"/>
2047         <xsd:any namespace="##other" processContents="lax" minOccurs="0" maxOccurs="unbounded"/>
2048       </xsd:sequence>
2049       <xsd:attribute name="id" use="required" type="xsd:ID"/>
2050       <xsd:attribute name="xlink:type" use="fixed" type="xsd:string" value="simple"/>
2051       <xsd:attribute name="xlink:href" use="required" type="xsd:uriReference"/>
2052       <xsd:attribute name="xlink:role" type="xsd:uriReference"/>
2053     </xsd:complexType>
2054   </xsd:element>
2055   <xsd:element name="Schema">
2056     <xsd:complexType>
2057       <xsd:attribute name="location" use="required" type="xsd:uriReference"/>
2058       <xsd:attribute name="version" type="xsd:string"/>
2059     </xsd:complexType>
2060   </xsd:element>
2061   <!--MESSAGE HEADER -->
2062   <xsd:element name="MessageHeader">
2063     <xsd:complexType>
2064       <xsd:sequence>
2065         <xsd:element ref="From"/>
2066         <xsd:element ref="To"/>
2067         <xsd:element ref="CPAId"/>
2068         <xsd:element ref="ConversationId"/>
2069         <xsd:element ref="Service"/>
2070         <xsd:element ref="Action"/>
2071         <xsd:element ref="MessageData"/>
2072         <xsd:element ref="QualityOfServiceInfo" minOccurs="0" maxOccurs="1"/>
2073         <xsd:element ref="Description" minOccurs="0" maxOccurs="unbounded"/>
2074         <xsd:element ref="SequenceNumber" minOccurs="0" maxOccurs="1"/>
2075       </xsd:sequence>
2076       <xsd:attribute name="version" use="fixed" type="xsd:string" value="98.0"/>
2077       <xsd:attribute ref="soap:mustUnderstand" use="required"/>
  
```

```

2078     </xsd:complexType>
2079 </xsd:element>
2080 <xsd:element name="CPAId" type="xsd:string"/>
2081 <xsd:element name="ConversationId" type="xsd:string"/>
2082 <xsd:element name="Service" type="xsd:string"/>
2083 <xsd:element name="Action" type="xsd:string"/>
2084 <xsd:element name="MessageData">
2085   <xsd:complexType>
2086     <xsd:sequence>
2087       <xsd:element ref="MessageId"/>
2088       <xsd:element ref="Timestamp"/>
2089       <xsd:element ref="RefToMessageId" minOccurs="0" maxOccurs="1"/>
2090       <xsd:element ref="TimeToLive" minOccurs="0" maxOccurs="1"/>
2091     </xsd:sequence>
2092   </xsd:complexType>
2093 </xsd:element>
2094 <xsd:element name="MessageId" type="xsd:string"/>
2095 <xsd:element name="TimeToLive" type="xsd:timeInstant"/>
2096 <xsd:element name="QualityOfServiceInfo">
2097   <xsd:complexType>
2098     <xsd:attribute name="deliverySemantics" use="default" value="BestEffort">
2099       <xsd:simpleType>
2100         <xsd:restriction base="xsd:NMTOKEN">
2101           <xsd:enumeration value="OnceAndOnlyOnce"/>
2102           <xsd:enumeration value="BestEffort"/>
2103         </xsd:restriction>
2104       </xsd:simpleType>
2105     </xsd:attribute>
2106     <xsd:attribute name="messageOrderSemantics" use="default" value="NotGuaranteed">
2107       <xsd:simpleType>
2108         <xsd:restriction base="xsd:NMTOKEN">
2109           <xsd:enumeration value="Guaranteed"/>
2110           <xsd:enumeration value="NotGuaranteed"/>
2111         </xsd:restriction>
2112       </xsd:simpleType>
2113     </xsd:attribute>
2114     <xsd:attribute name="deliveryReceiptRequested" use="default" value="None">
2115       <xsd:simpleType>
2116         <xsd:restriction base="xsd:NMTOKEN">
2117           <xsd:enumeration value="Signed"/>
2118           <xsd:enumeration value="Unsigned"/>
2119           <xsd:enumeration value="None"/>
2120         </xsd:restriction>
2121       </xsd:simpleType>
2122     </xsd:attribute>
2123   </xsd:complexType>
2124 </xsd:element>
2125 <!-- TRACE HEADER LIST -->
2126 <xsd:element name="TraceHeaderList">
2127   <xsd:complexType>
2128     <xsd:sequence>
2129       <xsd:element ref="TraceHeader" maxOccurs="unbounded"/>
2130     </xsd:sequence>
2131     <xsd:attribute name="id" type="xsd:ID"/>
2132     <xsd:attribute name="version" use="fixed" type="xsd:string" value="98.0"/>
2133     <xsd:attribute ref="soap:mustUnderstand" use="required"/>
2134   </xsd:complexType>
2135 </xsd:element>
2136 <xsd:element name="TraceHeader">
2137   <xsd:complexType>
2138     <xsd:sequence>
2139       <xsd:element ref="SenderURI"/>
2140       <xsd:element ref="ReceiverURI"/>
2141       <xsd:element ref="Timestamp"/>
2142       <xsd:any namespace="##other" processContents="lax" minOccurs="0" maxOccurs="unbounded"/>
2143     </xsd:sequence>
2144
2145   </xsd:complexType>
2146 </xsd:element>
2147 <xsd:element name="SenderURI" type="xsd:uriReference"/>
2148 <xsd:element name="ReceiverURI" type="xsd:uriReference"/>

```

```
2149 <xsd:element name="SequenceNumber" type="xsd:positiveInteger"/>
2150 <!-- ACKNOWLEDGEMENT -->
2151 <xsd:element name="Acknowledgment">
2152   <xsd:complexType>
2153     <xsd:sequence>
2154       <xsd:element ref="Timestamp"/>
2155       <xsd:element ref="From" minOccurs="0" maxOccurs="1"/>
2156     </xsd:sequence>
2157     <xsd:attribute name="id" type="xsd:ID"/>
2158     <xsd:attribute name="version" use="fixed" type="xsd:string" value="98.0"/>
2159     <xsd:attribute ref="soap:mustUnderstand" use="required"/>
2160     <xsd:attribute name="type" use="default" value="DeliveryReceipt">
2161       <xsd:simpleType>
2162         <xsd:restriction base="xsd:NMTOKEN">
2163           <xsd:enumeration value="DeliveryReceipt"/>
2164           <xsd:enumeration value="Acknowledgment"/>
2165         </xsd:restriction>
2166       </xsd:simpleType>
2167     </xsd:attribute>
2168     <xsd:attribute name="signed" type="xsd:boolean"/>
2169   </xsd:complexType>
2170 </xsd:element>
2171 <!-- ERROR LIST -->
2172 <xsd:element name="ErrorList">
2173   <xsd:complexType>
2174     <xsd:sequence>
2175       <xsd:element ref="Error" maxOccurs="unbounded"/>
2176     </xsd:sequence>
2177     <xsd:attribute name="id" type="xsd:ID"/>
2178     <xsd:attribute name="version" use="fixed" type="xsd:string" value="98.0"/>
2179     <xsd:attribute ref="soap:mustUnderstand" use="required"/>
2180     <xsd:attribute name="highestSeverity" use="default" value="Warning">
2181       <xsd:simpleType>
2182         <xsd:restriction base="xsd:string">
2183           <xsd:enumeration value="Warning"/>
2184           <xsd:enumeration value="Error"/>
2185         </xsd:restriction>
2186       </xsd:simpleType>
2187     </xsd:attribute>
2188   </xsd:complexType>
2189 </xsd:element>
2190 <xsd:element name="Error" type="xsd:string"/>
2191   <xsd:complexType>
2192     <xsd:attribute name="codeContext" use="required" type="xsd:uriReference"/>
2193     <xsd:attribute name="errorCode" use="required" type="xsd:string"/>
2194     <xsd:attribute name="severity" use="default" value="Warning">
2195       <xsd:simpleType>
2196         <xsd:restriction base="xsd:NMTOKEN">
2197           <xsd:enumeration value="Warning"/>
2198           <xsd:enumeration value="Error"/>
2199         </xsd:restriction>
2200       </xsd:simpleType>
2201     </xsd:attribute>
2202     <xsd:attribute name="location" type="xsd:string"/>
2203     <xsd:attribute name="xml:lang" type="xsd:language"/>
2204   </xsd:complexType> </xsd:element>
2205 <!-- STATUS DATA -->
2206 <xsd:element name="StatusData">
2207   <xsd:complexType>
2208     <xsd:sequence>
2209       <xsd:element ref="RefToMessageId"/>
2210       <xsd:element ref="Timestamp" minOccurs="0" maxOccurs="1"/>
2211     </xsd:sequence>
2212     <xsd:attribute name="version" use="fixed" type="xsd:string" value="98.0"/>
2213     <xsd:attribute ref="soap:mustUnderstand" use="required"/>
2214     <xsd:attribute name="messageStatus">
2215       <xsd:simpleType>
2216         <xsd:restriction base="xsd:NMTOKEN">
2217           <xsd:enumeration value="Unauthorized"/>
2218           <xsd:enumeration value="NotRecognized"/>
2219           <xsd:enumeration value="Received"/>
```

```

2220     </xsd:restriction>
2221     </xsd:simpleType>
2222     </xsd:attribute>
2223     </xsd:complexType>
2224 </xsd:element>
2225 <!-- COMMON ELEMENTS -->
2226 <xsd:element name="PartyId">
2227     <xsd:complexType>
2228         <xsd:simpleContent>
2229             <xsd:extension base="xsd:string">
2230                 <xsd:attribute name="type" type="xsd:string"/>
2231             </xsd:extension>
2232         </xsd:simpleContent>
2233     </xsd:complexType>
2234 </xsd:element>
2235 <xsd:element name="To">
2236     <xsd:complexType>
2237         <xsd:sequence>
2238             <xsd:element ref="PartyId"/>
2239         </xsd:sequence>
2240     </xsd:complexType>
2241 </xsd:element>
2242 <xsd:element name="From">
2243     <xsd:complexType>
2244         <xsd:sequence>
2245             <xsd:element ref="PartyId"/>
2246         </xsd:sequence>
2247     </xsd:complexType>
2248 </xsd:element>
2249 <xsd:element name="Description">
2250     <xsd:complexType>
2251         <xsd:simpleContent>
2252             <xsd:extension base="xsd:string">
2253                 <xsd:attribute name="xml:lang" type="xsd:NMTOKEN"/>
2254             </xsd:extension>
2255         </xsd:simpleContent>
2256     </xsd:complexType>
2257 </xsd:element>
2258 <xsd:element name="RefToMessageId" type="xsd:string"/>
2259 <xsd:element name="Timestamp" type="xsd:timeInstant"/>
2260 <!-- VIA -->
2261 <xsd:element name="Via">
2262     <xsd:complexType>
2263         <xsd:sequence>
2264             <xsd:element ref="CPAId" minOccurs="0"/>
2265             <xsd:element ref="Service" minOccurs="0"/>
2266             <xsd:element ref="Action" minOccurs="0"/>
2267         </xsd:sequence>
2268         <xsd:attribute name="version" use="required" type="xsd:string"/>
2269         <xsd:attribute ref="soap:mustUnderstand"/>
2270         <xsd:attribute ref="soap:actor"/>
2271         <xsd:attribute name="syncReply" type="xsd:boolean"/>
2272         <xsd:attribute name="deliveryReceiptRequested" use="default" value="None">
2273             <xsd:simpleType>
2274                 <xsd:restriction base="xsd:string">
2275                     <xsd:enumeration value="Signed"/>
2276                     <xsd:enumeration value="Unsigned"/>
2277                     <xsd:enumeration value="None"/>
2278                 </xsd:restriction>
2279             </xsd:simpleType>
2280         </xsd:attribute>
2281         <xsd:attribute name="reliableMessagingMethod">
2282             <xsd:simpleType>
2283                 <xsd:restriction base="xsd:string">
2284                     <xsd:enumeration value="ebXML"/>
2285                     <xsd:enumeration value="Transport"/>
2286                 </xsd:restriction>
2287             </xsd:simpleType>
2288         </xsd:attribute>
2289         <xsd:attribute name="ackRequested" type="xsd:boolean"/>
2290     </xsd:complexType>

```

2291
2292

```
</xsd:element>  
</xsd:schema>
```

2293 **Appendix B Communication Protocol Bindings**

2294 **B.1 Introduction**

2295 One of the goals of ebXML's Transport, Routing and Packaging team is to design a message
2296 handling service that is usable over a variety of network and application level communication
2297 protocols. These protocols serve as the "carrier" of ebXML Messages and provide the underlying
2298 services necessary to carry out a complete ebXML Message exchange between two parties.
2299 HTTP, FTP, Java Message Service (JMS) and SMTP are examples of application level
2300 communication protocols. TCP and SNA/LU6.2 are examples of network transport protocols.
2301 Communication protocols vary in their support for data content, processing behavior and error
2302 handling and reporting. For example, it is customary to send binary data in raw form over HTTP.
2303 However, in the case of SMTP it is customary to "encode" binary data into a 7-bit representation.
2304 HTTP is equally capable of carrying out synchronous or asynchronous message exchanges
2305 whereas it is likely that message exchanges occurring over SMTP will be asynchronous. This
2306 section describes the technical details needed to implement this abstract ebXML Message
2307 Handling Service over particular communication protocols.

2308 This section specifies communication protocol bindings and technical details for carrying *ebXML*
2309 *Message Service* messages for the following communication protocols:

- 2310 • Hypertext Transfer Protocol [HTTP], in both asynchronous and synchronous forms of
2311 transfer.
- 2312 • Simple Mail Transfer Protocol [SMTP], in asynchronous form of transfer only.

2313 **B.2 HTTP**

2314 **B.2.1 Minimum level of HTTP protocol**

2315 Hypertext Transfer Protocol Version 1.1 [HTTP] (<http://www.ietf.org/rfc2616.txt>) is the minimum
2316 level of protocol that MUST be used.

2317 **B.2.2 Sending ebXML Service messages over HTTP**

2318 Even though several HTTP request methods are available, this specification only defines the use
2319 of HTTP POST requests for sending *ebXML Message Service* messages over HTTP. The identity
2320 of the ebXML MSH (e.g. ebxmlhandler) may be part of the HTTP POST request:
2321 POST /ebxmlhandler HTTP/1.1
2322

2323 Prior to sending over HTTP, an ebXML Message MUST be formatted according to ebXML
2324 Message Service Specification sections 7 and 8. Additionally, the messages MUST conform to
2325 the HTTP specific MIME canonical form constraints specified in section 19.4 of RFC 2616 [HTTP]
2326 specification (see: <http://www.ietf.org/rfc2616.txt>).
2327

2328 HTTP protocol natively supports 8-bit and Binary data. Hence, transfer encoding is OPTIONAL
2329 for such parts in an ebXML Service Message prior to sending over HTTP. However, content-
2330 transfer-encoding of such parts (e.g. using base64 encoding scheme) is not precluded by this
2331 specification.
2332

2333 The rules for forming an HTTP message containing an ebXML Service Message are as follows:

- 2334 • The **Content-Type: Multipart/Related** MIME header with the associated
2335 parameters, from the ebXML Service Message Envelope **MUST** appear as an HTTP
2336 header.
- 2337 • All other MIME headers that constitute the ebXML Message Envelope **MUST** also
2338 become part of the HTTP header.
- 2339 • The mandatory SOAPAction HTTP header field must also be included in the HTTP
2340 header and must have a value of ebXML.

2341 SOAPAction: **ebXML**

- 2342 • Other headers with semantics defined by MIME specifications, such as Content-Transfer-
2343 Encoding, **SHALL NOT** appear as HTTP headers. Specifically, the "MIME-Version: 1.0"
2344 header **MUST NOT** appear as an HTTP header. However, HTTP-specific MIME-like
2345 headers defined by HTTP 1.1 **MAY** be used with the semantic defined in the HTTP
2346 specification.
- 2347 • All ebXML Service Message parts that follow the ebXML Message Envelope, including
2348 the MIME boundary string, constitute the HTTP entity body. This encompasses the SOAP
2349 envelope and the constituent ebXML parts and attachments including the trailing MIME
2350 boundary strings.

2351 The example below shows an example instance of an HTTP POST'ed ebXML Service Message:

```

2352 POST /servlet/ebXMLhandler HTTP/1.1
2353 Host: www.example2.com
2354 SOAPAction:
2355 Content-type: multipart/related; boundary="Boundary"; type="text/xml";
2356 start=" <ebxhmheader111@example.com>"
2357
2358 --Boundary
2359 Content-ID: <ebxhmheader111@example.com>
2360 Content-Type: text/xml
2361 <SOAP-ENV:Envelope xmlns:SOAP-ENV='http://schemas.xmlsoap.org/soap/envelope/'
2362 xmlns:eb='http://www.ebxml.org/namespaces/messageHeader'>
2363 <SOAP-ENV:Header>
2364 <eb:MessageHeader SOAP-ENV:mustUnderstand="1" eb:version="98.0">
2365 <eb:From>
2366 <eb:PartyId>urn:duns:123456789</eb:PartyId>
2367 </eb:From>
2368 <eb:To>
2369 <eb:PartyId>urn:duns:912345678</eb:PartyId>
2370 </eb:To>
2371 <eb:CPAId>20001209-133003-28572</eb:CPAId>
2372 <eb:ConversationId>20001209-133003-28572</eb:ConversationId>
2373 <eb:Service>OrderProcessing</eb:Service>
2374 <eb:Action>NewOrder</eb:Action>
2375 <eb:MessageData>
2376 <eb:MessageId>example.com.20001209-133003-28572</eb:MessageId>
2377 <eb:Timestamp>20010215111212Z</Timestamp>
2378 </eb:MessageData>
2379 <eb:QualityOfServiceInfo deliverySemantics="BestEffort"/>
2380 </eb:MessageHeader>
2381 </SOAP-ENV:Header>
2382 <SOAP-ENV:Body>
2383 <eb:Manifest SOAP-ENV:mustUnderstand="1" eb:version="98.0">
2384 <eb:Reference xlink:href="cid:ebxmlpayload111@example.com"
2385 xlink:role="XLinkRole"
2386 xlink:type="simple">
2387 <eb:Description xml:lang="en-us">Purchase Order 1</eb:Description>
2388 </eb:Reference>
2389 </eb:Manifest>
2390 </SOAP-ENV:Body>
2391 </SOAP-ENV:Envelope>
2392 --Boundary
2393 Content-ID: <ebxmlpayload111@example.com>
2394 Content-Type: text/xml
2395 <?xml version="1.0" encoding="UTF-8"?>

```

```
2397 <purchase_order>
2398   <po_number>1</po_number>
2399   <part_number>123</part_number>
2400   <price currency="USD">500.00</price>
2401 </purchase_order>
2402 --Boundary--
```

2403 B.2.3 HTTP Response Codes

2404 In general, semantics of communicating over HTTP as specified in the [RFC2616] MUST be
2405 followed, for returning the HTTP level response codes. A 2xx code MUST be returned when the
2406 HTTP Posted message is successfully received by the receiving HTTP entity. However, see
2407 exception for SOAP error conditions below. Similarly, other HTTP codes in the 3xx, 4xx, 5xx
2408 range MAY be returned for conditions corresponding to them. However, error conditions
2409 encountered while processing an ebXML Service Message MUST be reported using the error
2410 mechanism defined by the ebXML Message Service Specification (see section 11).

2411 B.2.4 SOAP Error conditions and Synchronous Exchanges

2412 The SOAP 1.1 specification states:

2413 *"In case of a SOAP error while processing the request, the SOAP HTTP server MUST issue an*
2414 *HTTP 500 "Internal Server Error" response and include a SOAP message in the response*
2415 *containing a SOAP Fault element indicating the SOAP processing error. "*

2416 However, the scope of the SOAP 1.1 specification is limited to synchronous mode of message
2417 exchange over HTTP, whereas the ebXML Message Service Specification specifies both
2418 synchronous and asynchronous modes of message exchange over HTTP. Hence, the SOAP 1.1
2419 specification MUST be followed for synchronous mode of message exchange, where the *SOAP*
2420 *Message* containing a SOAP Fault element indicating the SOAP processing error MUST be
2421 returned in the HTTP response with a response code of "HTTP 500 Internal Server Error". When
2422 asynchronous mode of message exchange is being used, a HTTP response code in the range
2423 2xx MUST be returned when the message is received successfully and any error conditions
2424 (including SOAP errors) must be returned via a separate HTTP Post.

2425 B.2.5 Synchronous vs. Asynchronous

2426 When the **syncReply** parameter in the **Via** element is set to "true", the response message(s)
2427 MUST be returned on the same HTTP connection as the inbound request, with an appropriate
2428 HTTP response code, as described above. When the **syncReply** parameter in the ebXML
2429 Header is set to "false", the response messages are not returned on the same HTTP connection
2430 as the inbound request, but using an independent HTTP Post request. An HTTP response with a
2431 response code as defined in section B.2.3 above and with an empty HTTP body MUST be
2432 returned in response to the HTTP Post, however.

2433 B.2.6 Access Control

2434 Implementers MAY protect their ebXML Message Service Handlers from unauthorized access
2435 through the use of an access control mechanism. The HTTP access authentication process
2436 described in "HTTP Authentication: Basic and Digest Access Authentication" [RFC2617] defines
2437 the access control mechanisms allowed to protect an ebXML Message Service Handler from
2438 unauthorized access.

2439 Implementers MAY support all of the access control schemes defined in [RFC2617] however they
2440 MUST support the Basic Authentication mechanism, as described in section 2, when Access
2441 Control is used.

2442 Implementers that use basic authentication for access control SHOULD also use communication
2443 protocol level security, as specified in the section titled "Confidentiality and Communication
2444 Protocol Level Security" in this document.

2445 **B.2.7 Confidentiality and Communication Protocol Level Security**

2446 An ebXML Message Service Handler MAY use transport layer encryption to protect the
2447 confidentiality of ebXML Messages and HTTP transport headers. The IETF Transport Layer
2448 Security specification [RFC2246] provides the specific technical details and list of allowable
2449 options, which may be used by ebXML Message Service Handlers. ebXML Message Service
2450 Handlers MUST be capable of operating in backwards compatibility mode with SSL [SSL3], as
2451 defined in Appendix E of [RFC2246].

2452 ebXML Message Service Handlers MAY use any of the allowable encryption algorithms and key
2453 sizes specified within [RFC2246]. At a minimum ebXML Message Service Handlers MUST
2454 support the key sizes and algorithms necessary for backward compatibility with [SSL3].

2455 The use of 40-bit encryption keys/algorithms is permitted, however it is RECOMMENDED that
2456 stronger encryption keys/algorithms SHOULD be used.

2457 Both [RFC2246] and [SSL3] require the use of server side digital certificates. In addition client
2458 side certificate based authentication is also permitted. ebXML Message Service handlers MUST
2459 support 3rd party signed certificates as well as "self signed" certificates.

2460 **B.3 SMTP**

2461 The Simple Mail Transfer Protocol [SMTP] and its companion documents [RFC822] and [ESMTP]
2462 makeup the suite of specifications commonly referred to as Internet Electronic Mail. These
2463 specifications have been augmented over the years by other specifications, which define
2464 additional functionality "layered on top" of these baseline specifications. These include:

- 2465 • Multipurpose Internet Mail Extensions (MIME) [RFC2045], [RFC2046], [RFC2387]
- 2466 • SMTP Service Extension for Authentication [RFC2554]
- 2467 • SMTP Service Extension for Secure SMTP over TLS [RFC2487]

2468

2469 Typically, Internet Electronic Mail Implementations consist of two "agent" types:

- 2470 • Message Transfer Agent (MTA): Programs that send and receive mail messages with
2471 other MTA's on behalf of MUA's. Microsoft Exchange Server is an example of a MTA
- 2472 • Mail User Agent (MUA): Electronic Mail programs are used to construct electronic mail
2473 messages and communicate with an MTA to send/retrieve mail messages. Microsoft
2474 Outlook is an example of a MUA.

2475 MTA's often serve as "mail hubs" and can typically service hundreds or more MUA's.

2476

2477 MUA's are responsible for constructing electronic mail messages in accordance with the Internet
2478 Electronic Mail Specifications identified above. This section describes the "binding" of an ebXML
2479 compliant message for transport via eMail from the perspective of a MUA. No attempt is made to
2480 define the binding of an ebXML Message exchange over SMTP from the standpoint of a MTA.

2481 B.3.1 Minimum level of supported protocols

- 2482 • Simple Mail Transfer Protocol [RFC821] and [RFC822]
- 2483 • MIME [RFC2045] and [RFC2046]
- 2484 • Multipart/Related MIME [RFC2387]

2485

2486 B.3.2 Sending ebXML Messages over SMTP

2487

2488 Prior to sending messages over SMTP an ebXML Message MUST be formatted according to
2489 ebXML Message Service Specification sections 7 and 8. Additionally the messages must also
2490 conform to the syntax, format and encoding rules specified by MIME [RFC2045], [RFC2046] and
2491 [RFC2387].

2492 Many types of data that a party might desire to transport via email are represented as 8bit
2493 characters or binary data. Such data cannot be transmitted over SMTP [SMTP], which restricts
2494 mail messages to 7bit US-ASCII data with lines no longer than 1000 characters including any
2495 trailing CRLF line separator. If a sending Message Service Handler knows that a receiving MTA,
2496 or ANY intermediary MTA's, are restricted to handling 7-bit data then any ebXML header or
2497 payload data that uses 8 bit (or binary) representation must be "transformed" according to the
2498 encoding rules specified in section 6 of [RFC2045]. In cases where a Message Service Handler
2499 knows that a receiving MTA and ALL intermediary MTA's are capable of handling 8-bit data then
2500 no transformation is needed on any part of the ebXML Message.
2501

2502 The rules for forming an ebXML Message for transport via SMTP are as follows:

- 2503 • If using [RFC821] restricted transport paths, apply transfer encoding to all 8-bit data that
2504 will be transported in a ebXML header or payload body part, according to the encoding
2505 rules defined in section 6 of [RFC2045]. The Content-Transfer-Encoding MIME header
2506 MUST be included in the MIME envelope portion of any body part that has been
2507 transformed (encoded).
- 2508 • The Content-Type: Multipart/Related MIME header with the associated
2509 parameters, from the ebXML Message Envelope MUST appear as an eMail MIME
2510 header.
- 2511 • All other MIME headers that constitute the ebXML Message Envelope MUST also
2512 become part of the eMail MIME header.
- 2513 • The mandatory SOAPAction MIME header field must also be included in the eMail MIME
2514 header and must have the value of ebXML:

2515 SOAPAction: **ebXML**

2516 Where Service and Action are values of the corresponding elements from
2517 the ebXML MessageHeader.

- 2518 • The "MIME-Version: 1.0" header must appear as an eMail MIME header.
- 2519 • The eMail header "To:" MUST contain the [RFC822] compliant eMail address of the
2520 ebXML Message Service Handler.
- 2521 • The eMail header "From:" MUST contain the [RFC822] compliant eMail address of the
2522 senders ebXML Message Service handler.
- 2523 • Construct a "Date:" eMail header in accordance with [RFC822]

- 2524 • Other headers MAY occur within the eMail message header in accordance with [RFC822]
 2525 and [RFC2045], however ebXML Message Service Handlers MAY choose to ignore
 2526 them.

2527 The example below shows a minimal example of an eMail message containing an ebXML
 2528 Message:

```

2529 From: ebXMLhandler@example.com
2530 To: ebXMLhandler@example2.com
2531 Date: Thu, 08 Feb 2001 19:32:11 CST
2532 MIME-Version: 1.0
2533 SOAPAction:
2534 Content-type: multipart/related; boundary="Boundary"; type="text/xml";
2535 start=" <ebxhheader111@example.com>"
2536
2537 --Boundary
2538 Content-ID: <ebxhheader111@example.com>
2539 Content-Type: text/xml
2540
2541 <SOAP-ENV:Envelope xmlns:SOAP-ENV='http://schemas.xmlsoap.org/soap/envelope/'
2542 xmlns:eb='http://www.ebxml.org/namespaces/messageHeader'>
2543 <SOAP-ENV:Header>
2544 <eb:MessageHeader SOAP-ENV:mustUnderstand="1" eb:version="98.0">
2545 <eb:From>
2546 <eb:PartyId>urn:duns:123456789</eb:PartyId>
2547 </eb:From>
2548 <eb:To>
2549 <eb:PartyId>urn:duns:912345678</eb:PartyId>
2550 </eb:To>
2551 <eb:CPAId>20001209-133003-28572</eb:CPAId>
2552 <eb:ConversationId>20001209-133003-28572</eb:ConversationId>
2553 <eb:Service>OrderProcessing</eb:Service>
2554 <eb:Action>NewOrder</eb:Action>
2555 <eb:MessageData>
2556 <eb:MessageId>example.com.20001209-133003-28572</eb:MessageId>
2557 <eb:Timestamp>20010215111212Z</Timestamp>
2558 </eb:MessageData>
2559 <eb:QualityOfServiceInfo deliverySemantics="BestEffort"/>
2560 </eb:MessageHeader>
2561 </SOAP-ENV:Header>
2562 <SOAP-ENV:Body>
2563 <eb:Manifest SOAP-ENV:mustUnderstand="1" eb:version="98.0">
2564 <eb:Reference xlink:href="cid:ebxmlpayload111@example.com"
2565 xlink:role="XLinkRole"
2566 xlink:type="simple">
2567 <eb:Description xml:lang="en-us">Purchase Order 1</eb:Description>
2568 </eb:Reference>
2569 </eb:Manifest>
2570 </SOAP-ENV:Body>
2571 </SOAP-ENV:Envelope>
2572 --Boundary
2573 Content-ID: <ebxhheader111@example.com>
2574 Content-Type: text/xml
2575 <?xml version="1.0" encoding="UTF-8"?>
2576 <purchase_order>
2577 <po_number>1</po_number>
2578 <part_number>123</part_number>
2579 <price currency="USD">500.00</price>
2580 </purchase_order>
2581 --Boundary--
  
```

2583 B.3.3 Response Messages

2584 All ebXML response messages, including errors and acknowledgements, are delivered
 2585 asynchronously between ebXML Message Service Handlers. Each response message MUST be

2586 constructed in accordance with the rules specified in the section titled "Sending ebXML
2587 messages over SMTP" elsewhere in this document.

2588 ebXML Message Service Handlers **MUST** be capable of receiving a delivery failure notification
2589 message sent by an MTA. An MSH that receives a delivery failure notification message
2590 **SHOULD** examine the message to determine which ebXML message, sent by the MSH, resulted
2591 in a message delivery failure. The MSH **SHOULD** attempt to identify the application responsible
2592 for sending the offending message that caused the failure. The MSH **SHOULD** attempt to notify
2593 the application that a message delivery failure has occurred. If the MSH is unable to determine
2594 the source of the offending message the MSH administrator should be notified.

2595 MSH's which cannot identify a received message as a valid ebXML message or a message
2596 delivery failure **SHOULD** retain the unidentified message in a "dead letter" folder.

2597 A MSH **SHOULD** place an entry in an audit log indicating the disposition of each received
2598 message.

2599 **B.3.4 Access Control**

2600 Implementers **MAY** protect their ebXML Message Service Handlers from unauthorized access
2601 through the use of an access control mechanism. The SMTP access authentication process
2602 described in "SMTP Service Extension for Authentication" [RFC2554] defines the ebXML
2603 recommended access control mechanism to protect a SMTP based ebXML Message Service
2604 Handler from unauthorized access.

2605 **B.3.5 Confidentiality and Communication Protocol Level Security**

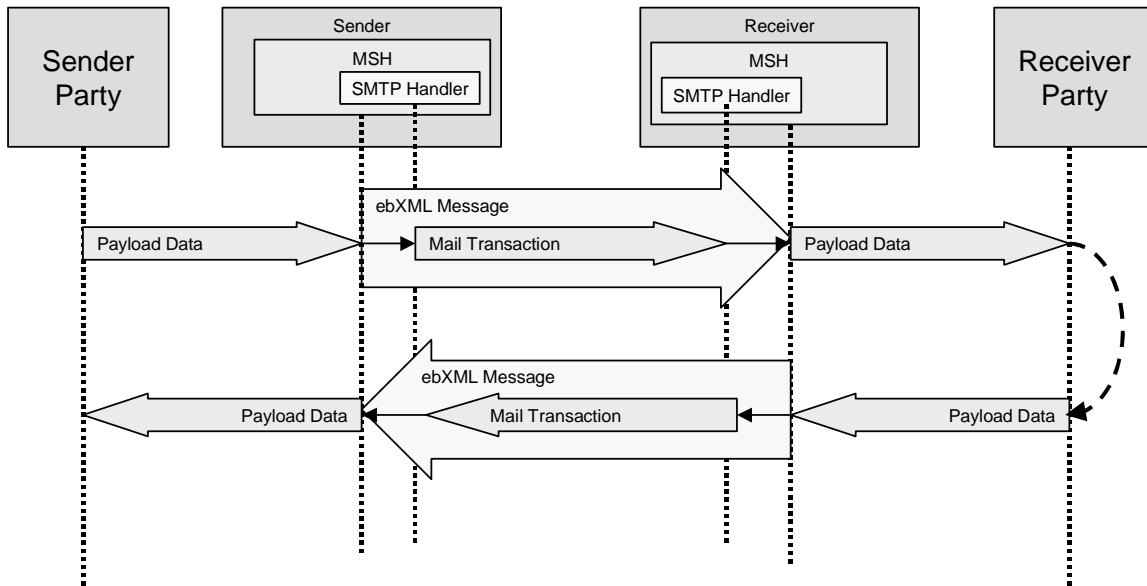
2606

2607 An ebXML Message Service Handler **MAY** use transport layer encryption to protect the
2608 confidentiality of ebXML messages. The IETF "SMTP Service Extension for Secure SMTP over
2609 TLS" specification [RFC2487] provides the specific technical details and list of allowable options,
2610 which may be used.

2611 **B.3.6 SMTP Model**

2612 All *ebXML Message Service* messages carried as mail in an [SMTP] Mail Transaction as shown
2613 in the figure below.

2614



2615

2616

2617 **B.4 Communication Errors during Reliable Messaging**

2618 When the Sender or the Receiver detects a transport protocol level error (such as an HTTP,
 2619 SMTP or FTP error) and Reliable Messaging is being used then the appropriate transport
 2620 recovery handler will execute a recovery sequence. Only if the error is unrecoverable, does
 2621 Reliable Messaging recovery take place (see section 10).

2622

2623 Copyright Statement

2624 Copyright © ebXML 2001. All Rights Reserved.

2625

2626 This document and translations of it MAY be copied and furnished to others, and derivative works
2627 that comment on or otherwise explain it or assist in its implementation MAY be prepared, copied,
2628 published and distributed, in whole or in part, without restriction of any kind, provided that the
2629 above copyright notice and this paragraph are included on all such copies and derivative works.
2630 However, this document itself MAY not be modified in any way, such as by removing the
2631 copyright notice or references to the ebXML, UN/CEFACT, or OASIS, except as required to
2632 translate it into languages other than English.

2633 The limited permissions granted above are perpetual and will not be revoked by ebXML or its
2634 successors or assigns.

2635 This document and the information contained herein is provided on an "AS IS" basis and ebXML
2636 disclaims all warranties, express or implied, including but not limited to any warranty that the use
2637 of the information herein will not infringe any rights or any implied warranties of merchantability or
2638 fitness for a particular purpose.

2639