



Creating A Single Global Electronic Market

1

Message Service Specification

2

ebXML Transport, Routing & Packaging

3

Version 1.0

4

11 May 2001

5 **1 Status of this Document**

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

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

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

12

13 *Specification*

14 This Technical Specification document has been approved by the ebXML Plenary.

15 This material fulfils requirements of the ebXML Requirements document.

16

17 This version

18 <http://www.ebxml.org/specs/ebMS.pdf>

19 Latest version

20 <http://www.ebxml.org/specs/ebMS.pdf>

21

22 **2 ebXML Participants**

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

26

27

28	Ralph Berwanger – bTrade.com	49	Ravi Kacker – Kraft Foods
29	Jonathan Borden – Author of XMTP	50	Henry Lowe – OMG
30	Jon Bosak – Sun Microsystems	51	Jim McCarthy – webXI
31	Marc Breissinger – webMethods	52	Bob Miller – GXS
32	Dick Brooks – Group 8760	53	Dale Moberg – Sterling Commerce
33	Doug Bunting – Ariba	54	Joel Munter – Intel
34	David Burdett – Commerce One	55	Shumpei Nakagaki – NEC Corporation
35	David Craft – VerticalNet	56	Farrukh Najmi – Sun Microsystems
36	Philippe De Smedt – Viquity	57	Akira Ochi – Fujitsu
37	Lawrence Ding – WorldSpan	58	Martin Sachs, IBM
38	Rik Drummond – Drummond Group	59	Saikat Saha – Commerce One
39	Andrew Eisenberg – Progress Software	60	Masayoshi Shimamura – Fujitsu
40	Colleen Evans –Progress / Sonic Software	61	Prakash Sinha – Netfish Technologies
41	David Fischer – Drummond Group	62	Rich Salz – Zolera Systems
42	Christopher Ferris – Sun Microsystems	63	Tae Joon Song – eSum Technologies, Inc.
43	Robert Fox – Softshare	64	Kathy Spector – Extricity
44	Brian Gibb – Sterling Commerce	65	Nikola Stojanovic – Encoda Systems, Inc.
45	Maryann Hondo – IBM	66	David Turner - Microsoft
46	Jim Hughes – Fujitsu	67	Gordon Van Huizen – Progress Software
47	John Ibbotson – IBM	68	Martha Warfelt – DaimlerChrysler Corporation
48	Ian Jones – British Telecommunications	69	Prasad Yendluri – Web Methods

70 **3 Table of Contents**

71 1 Status of this Document 2

72 2 ebXML Participants 2

73 3 Table of Contents 4

74 4 Introduction..... 8

75 4.1 Summary of Contents of Document..... 8

76 4.2 Document Conventions 8

77 4.3 Audience..... 9

78 4.4 Caveats and Assumptions 9

79 4.5 Related Documents 9

80 5 Design Objectives 10

81 6 System Overview 11

82 6.1 Message Service Purpose..... 11

83 6.2 Message Service Overview 11

84 6.3 Use of version attribute 12

85 7 Packaging Specification..... 13

86 7.1 Introduction 13

87 7.1.1 SOAP Structural Conformance..... 14

88 7.2 Message Package 14

89 7.3 Header Container 14

90 7.3.1 Content-Type..... 14

91 7.3.2 Header Container Example..... 15

92 7.4 Payload Container 15

93 7.4.1 Example of a Payload Container..... 15

94 7.5 Additional MIME Parameters 15

95 7.6 Reporting MIME Errors..... 15

96 8 ebXML SOAP Extensions 16

97 8.1 XML Prolog 16

98 8.1.1 XML Declaration 16

99 8.1.2 Encoding Declaration..... 16

100 8.2 ebXML SOAP Envelope extensions 16

101 8.2.1 Namespace pseudo attribute 16

102 8.2.2 xsi:schemaLocation attribute 17

103 8.2.3 ebXML SOAP Extensions 18

104 8.2.4 #wildcard element content..... 18

105 8.2.5 id attributes 18

106 8.3 SOAP Header element 18

107 8.4 MessageHeader element 19

108 8.4.1 From and To elements..... 19

109 8.4.2 CPAlid element..... 20

110 8.4.3 ConversationId element..... 20

111 8.4.4 Service element..... 21

112 8.4.5 Action element..... 21

113 8.4.6 MessageData element..... 21

114 8.4.7 QualityOfServiceInfo element..... 22

115	8.4.8	SequenceNumber element.....	23
116	8.4.9	Description element.....	24
117	8.4.10	version attribute.....	24
118	8.4.11	SOAP mustUnderstand attribute.....	24
119	8.4.12	MessageHeader Sample.....	25
120	8.5	TraceHeaderList element.....	25
121	8.5.1	SOAP actor attribute.....	25
122	8.5.2	TraceHeader element.....	25
123	8.5.3	Single Hop TraceHeader Sample.....	27
124	8.5.4	Multi-hop TraceHeader Sample.....	28
125	8.6	Acknowledgment Element.....	29
126	8.6.1	Timestamp element.....	29
127	8.6.2	From element.....	29
128	8.6.3	ds:Reference element.....	30
129	8.6.4	SOAP actor attribute.....	30
130	8.6.5	Acknowledgement Sample.....	30
131	8.7	Via element.....	30
132	8.7.1	SOAP mustUnderstand attribute.....	30
133	8.7.2	SOAP actor attribute.....	31
134	8.7.3	syncReply attribute.....	31
135	8.7.4	reliableMessagingMethod attribute.....	31
136	8.7.5	ackRequested attribute.....	31
137	8.7.6	CPAId element.....	31
138	8.7.7	Service and Action elements.....	31
139	8.7.8	Via element Sample.....	32
140	8.8	ErrorList element.....	32
141	8.8.1	id attribute.....	32
142	8.8.2	highestSeverity attribute.....	32
143	8.8.3	Error element.....	32
144	8.8.4	ErrorList Sample.....	33
145	8.8.5	errorCode values.....	33
146	8.9	Signature element.....	34
147	8.10	SOAP Body Extensions.....	35
148	8.11	Manifest element.....	35
149	8.11.1	id attribute.....	35
150	8.11.2	#wildcard element.....	35
151	8.11.3	Reference element.....	35
152	8.11.4	References included in a Manifest.....	36
153	8.11.5	Manifest Validation.....	36
154	8.11.6	Manifest Sample.....	37
155	8.12	StatusRequest Element.....	37
156	8.12.1	StatusRequest Sample.....	37
157	8.13	StatusResponse element.....	37
158	8.13.1	RefToMessageId element.....	37
159	8.13.2	Timestamp element.....	37
160	8.13.3	messageStatus attribute.....	37
161	8.13.4	StatusResponse Sample.....	38
162	8.14	DeliveryReceipt element.....	38
163	8.14.1	Timestamp element.....	38
164	8.14.2	ds:Reference element.....	38
165	8.14.3	DeliveryReceipt Sample.....	38
166	8.15	Combining ebXML SOAP Extension Elements.....	39
167	8.15.1	Manifest element.....	39
168	8.15.2	MessageHeader element.....	39
169	8.15.3	TraceHeaderList element.....	39
170	8.15.4	StatusRequest element.....	39
171	8.15.5	StatusResponse element.....	39
172	8.15.6	ErrorList element.....	39

173 8.15.7 Acknowledgment element.....39

174 8.15.8 Delivery Receipt element.....39

175 8.15.9 Signature element.....39

176 8.15.10 Via element.....39

177 **9 Message Service Handler Services40**

178 9.1 Message Status Request Service40

179 9.1.1 Message Status Request Message.....40

180 9.1.2 Message Status Response Message.....40

181 9.1.3 Security Considerations.....41

182 9.2 Message Service Handler Ping Service.....41

183 9.2.1 Message Service Handler Ping Message.....41

184 9.2.2 Message Service Handler Pong Message.....41

185 9.2.3 Security Considerations.....42

186 **10 Reliable Messaging43**

187 10.1.1 Persistent Storage and System Failure.....43

188 10.1.2 Methods of Implementing Reliable Messaging.....43

189 10.2 Reliable Messaging Parameters43

190 10.2.1 Delivery Semantics.....43

191 10.2.2 mshTimeAccuracy.....44

192 10.2.3 TimeToLive.....44

193 10.2.4 reliableMessagingMethod.....44

194 10.2.5 ackRequested.....44

195 10.2.6 retries.....44

196 10.2.7 retryInterval.....45

197 10.2.8 persistDuration.....45

198 10.3 ebXML Reliable Messaging Protocol.....45

199 10.3.1 Sending Message Behavior.....45

200 10.3.2 Receiving Message Behavior.....46

201 10.3.3 Generating an Acknowledgement Message.....46

202 10.3.4 Resending Lost Messages and Duplicate Filtering.....47

203 10.3.5 Duplicate Message Handling.....48

204 10.4 Failed Message Delivery49

205 **11 Error Reporting and Handling.....50**

206 11.1 Definitions50

207 11.2 Types of Errors50

208 11.3 When to generate Error Messages50

209 11.3.1 Security Considerations.....51

210 11.4 Identifying the Error Reporting Location.....51

211 11.5 Service and Action Element Values.....51

212 **12 Security.....52**

213 12.1 Security and Management.....52

214 12.2 Collaboration Protocol Agreement.....52

215 12.3 Countermeasure Technologies52

216 12.3.1 Persistent Digital Signature.....52

217 12.3.2 Persistent Signed Receipt.....54

218 12.3.3 Non-persistent Authentication.....54

219 12.3.4 Non-persistent Integrity.....55

220 12.3.5 Persistent Confidentiality.....55

221 12.3.6 Non-persistent Confidentiality.....55

222 12.3.7 Persistent Authorization.....55

223 12.3.8 Non-persistent Authorization.....55

224 12.3.9 Trusted Timestamp.....55

225 12.3.10 Supported Security Services.....55

226 13 References58

227 13.1 Normative References58

228 13.2 Non-Normative References59

229 14 Contact Information.....60

230 Appendix A ebXML SOAP Extension Elements Schema62

231 Appendix B Communication Protocol Bindings.....68

232 B.1 Introduction68

233 B.2 HTTP68

234 B.2.1 Minimum level of HTTP protocol68

235 B.2.2 Sending ebXML Service messages over HTTP68

236 B.2.3 HTTP Response Codes70

237 B.2.4 SOAP Error conditions and Synchronous Exchanges70

238 B.2.5 Synchronous vs. Asynchronous70

239 B.2.6 Access Control70

240 B.2.7 Confidentiality and Communication Protocol Level Security71

241 B.3 SMTP71

242 B.3.1 Minimum level of supported protocols71

243 B.3.2 Sending ebXML Messages over SMTP72

244 B.3.3 Response Messages73

245 B.3.4 Access Control74

246 B.3.5 Confidentiality and Communication Protocol Level Security74

247 B.3.6 SMTP Model74

248 B.4 Communication Errors during Reliable Messaging74

249 Disclaimer75

250 Copyright Statement75

251

251 4 Introduction

252 This specification is one of a series of specifications that realize the vision of creating a single global
253 electronic marketplace where enterprises of any size and in any geographical location can meet and
254 conduct business with each other through the exchange of XML based messages. The set of
255 specifications enable a modular, yet complete electronic business framework.

256 This specification focuses on defining a communications-protocol neutral method for exchanging the
257 electronic business messages. It defines specific enveloping constructs that support reliable, secure
258 delivery of business information. Furthermore, the specification defines a flexible enveloping technique
259 that permits ebXML-compliant messages to contain payloads of any format type. This versatility ensures
260 that legacy electronic business systems employing traditional syntaxes (i.e. UN/EDIFACT, ASC X12, or
261 HL7) can leverage the advantages of the ebXML infrastructure along with users of emerging technologies

262 4.1 Summary of Contents of Document

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

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

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

270 This specification is organized around the following topics:

- 271 • **Packaging Specification** – A description of how to package an ebXML Message and its associated
272 parts into a form that can sent using a communications protocol such as HTTP or SMTP (section 7)
- 273 • **ebXML SOAP Extensions** – A specification of the structure and composition of the information
274 necessary for an *ebXML Message Service* to successfully generate or process an ebXML Message
275 (section 8)
- 276 • **Message Service Handler Services** – A description of two services that enable one service to
277 discover the status of another Message Service Handler (MSH) or an individual message (section 9)
- 278 • **Reliable Messaging** – The Reliable Messaging function defines an interoperable protocol such that
279 any two Message Service implementations can “reliably” exchange messages that are sent using
280 “reliable messaging” once-and-only-once delivery semantics (section 10)
- 281 • **Error Handling** – This section describes how one *ebXML Message Service* reports errors it detects
282 to another ebXML Message Service Handler (section 11)
- 283 • **Security** – This provides a specification of the security semantics for ebXML Messages (section12).

284 Appendices to this specification cover the following:

- 285 • **Appendix A Schema** – This normative appendix contains [XMLSchema] for the ebXML SOAP
286 *Header* and *Body*.
- 287 • **Appendix B Communication Protocol Envelope Mappings** – This normative appendix describes
288 how to transport *ebXML Message Service* compliant messages over [HTTP] and [SMTP]

289 4.2 Document Conventions

290 Terms in *Italics* are defined in the ebXML Glossary of Terms [ebGLOSS]. Terms listed in **Bold Italics**
291 represent the element and/or attribute content. Terms listed in *Courier* font relate to MIME
292 components. Notes are listed in Times New Roman font and are informative (non-normative).

293 The keywords MUST, MUST NOT, REQUIRED, SHALL, SHALL NOT, SHOULD, SHOULD NOT,
294 RECOMMENDED, MAY, and OPTIONAL, when they appear in this document, are to be interpreted as
295 described in [RFC2119] as quoted here:

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

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

317 **4.3 Audience**

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

320 **4.4 Caveats and Assumptions**

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

323 All examples are to be considered non-normative. If inconsistencies exist between the specification and
324 the examples, the specification supersedes the examples.

325 **4.5 Related Documents**

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

- 328 • **ebXML Message Services Requirements Specification**[ebMSREQ] – defines the requirements of
329 these Message Services
- 330 • **ebXML Technical Architecture Specification**[ebTA] – defines the overall technical architecture for
331 ebXML
- 332 • **ebXML Technical Architecture Security Specification**[ebTASEC] – defines the security
333 mechanisms necessary to negate anticipated, selected threats
- 334 • **ebXML Collaboration Protocol Profile and Agreement Specification**[ebCPP] - defines how one
335 party can discover and/or agree upon the information that party needs to know about another party
336 prior to sending them a message that complies with this specification
- 337 • **ebXML Registry/Repository Services Specification**[ebRS] – defines a registry service for the
338 ebXML environment

339 **5 Design Objectives**

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

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

351 6 System Overview

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

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

362 6.1 Message Service Purpose

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

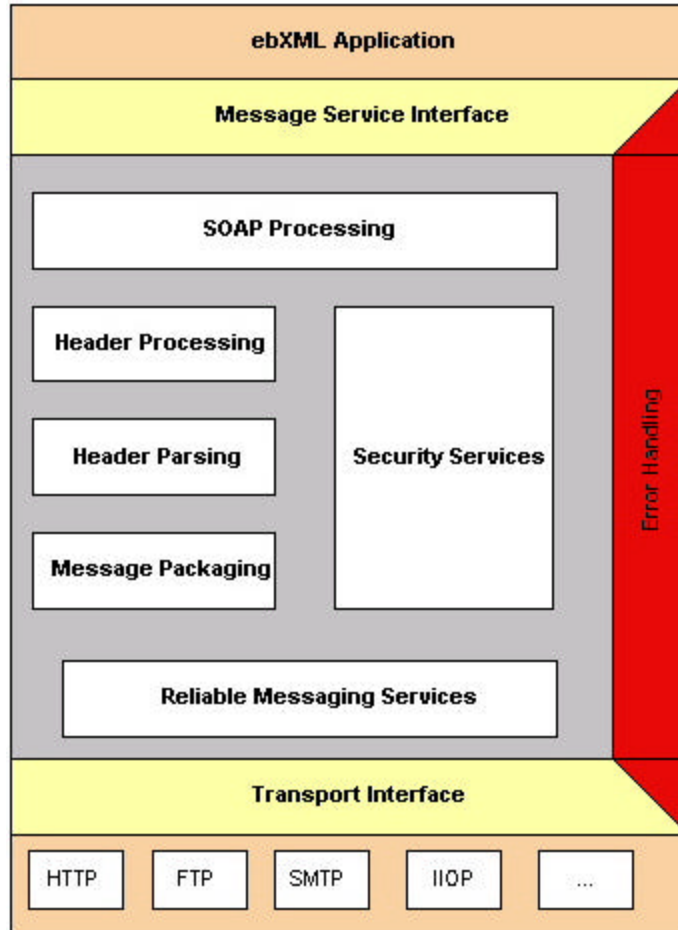
366 The *ebXML Message Service* is provided for environments requiring a robust, yet low cost solution to
367 enable electronic business. It is one of the four "infrastructure" components of ebXML. The other three
368 are: Registry/Repository [ebRS], Collaboration Protocol Profile/Agreement [ebCPP] and ebXML
369 Technical Architecture [ebTA].

370 6.2 Message Service Overview

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

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

- 377 • **Header Processing** - the creation of the SOAP **Header** elements for the *ebXML Message* uses input
378 from the application, passed through the Message Service Interface, information from the
379 *Collaboration Protocol Agreement (CPA)* defined in [ebCPP] that governs the message, and
380 generated information such as digital signature, timestamps and unique identifiers.
- 381 • **Header Parsing** - extracting or transforming information from a received SOAP **Header** or **Body**
382 element into a form that is suitable for processing by the MSH implementation.
- 383 • **Security Services** - digital signature creation and verification, authentication and authorization.
384 These services MAY be used by other components of the MSH including the Header Processing and
385 Header Parsing components.
- 386 • **Reliable Messaging Services** - handles the delivery and acknowledgment of ebXML Messages sent
387 with *deliverySemantics* of **OnceAndOnlyOnce**. The service includes handling for persistence,
388 retry, error notification and acknowledgment of messages requiring reliable delivery.
- 389 • **Message Packaging** - the final enveloping of an *ebXML Message* (SOAP **Header** or **Body** elements
390 and payload) into its SOAP Messages with Attachments [SOAPATTACH] container.
- 391 • **Error Handling** - this component handles the reporting of errors encountered during MSH or
392 Application processing of a message.
- 393 • **Message Service Interface** - an abstract service interface that applications use to interact with the
394 MSH to send and receive messages and which the MSH uses to interface with applications that
395 handle received messages.



396

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

398 **6.3 Use of version attribute**

399 Each ebXML SOAP extension element has its own version attribute, with a value that matches the
 400 ebXML Message Service Specification version level, to allow for elements to change in semantic meaning
 401 individually without changing the entire specification.

402 Use of multiple versions of ebXML SOAP extensions elements within the same ebXML SOAP document,
 403 while supported, should only be used in extreme cases where it becomes necessary to semantically
 404 change an element, which cannot wait for the next ebXML Message Service Specification version
 405 release.

406 7 Packaging Specification

407 7.1 Introduction

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

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

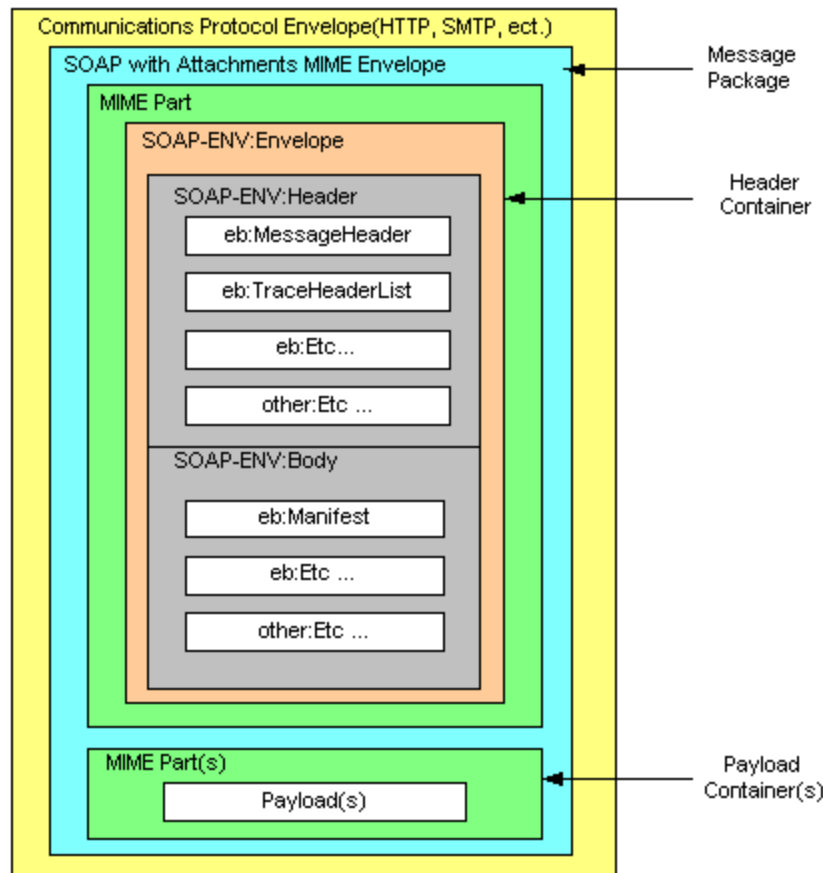
- 412 • A MIME part, referred to as the *Header Container*, containing one SOAP 1.1 compliant message.
 413 This XML document is referred to as a *SOAP Message* for the remainder of this specification,
- 414 • zero or more MIME parts, referred to as *Payload Containers*, containing application level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

415 The *SOAP Message* is an XML document that consists of the SOAP **Envelope** element. This is the root
 416 element of the XML document representing the *SOAP Message*. The SOAP **Envelope** element consists
 417 of the following:

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

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

423



424

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

426 **7.1.1 SOAP Structural Conformance**

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

- 428 • Simple Object Access Protocol (SOAP) 1.1 [SOAP]
- 429 • SOAP Messages with Attachments [SOAPATTACH]

430 Carrying ebXML headers in *SOAP Messages* does not mean that ebXML overrides existing semantics of
 431 SOAP, but rather that the semantics of ebXML over SOAP maps directly onto SOAP semantics.

432 **7.2 Message Package**

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

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

442
 443
 444
 445
 446
 447

```
Content-Type: multipart/related; type="text/xml"; boundary="boundaryValue";
start=messagepackage-123@example.com

--boundaryValue
Content-ID: messagepackage-123@example.com
```

448 **7.3 Header Container**

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

452 **7.3.1 Content-Type**

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

456
 457

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

458 **7.3.1.1 charset Attribute**

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

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

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

468
 469

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

470 **7.3.2 Header Container Example**

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

```

472
473 Content-ID: messagepackage-123@example.com          --- | Header
474 Content-Type: text/xml;                            --- |
475     charset="UTF-8"                                --- |
476
477 <SOAP-ENV:Envelope                                 -- | SOAP Message
478     xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
479   <SOAP-ENV:Header>
480     ...
481   </SOAP-ENV:Header>
482   <SOAP-ENV:Body>
483     ...
484   </SOAP-ENV:Body>
485 </SOAP-ENV:Envelope>                               -- |
486 ---boundaryValue                                   --- |
    
```

487 **7.4 Payload Container**

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

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

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

494 The contents of each *Payload Container* MUST be identified by the ebXML Message **Manifest** element
 495 within the SOAP **Body** (see section 8.11).

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

501 **7.4.1 Example of a Payload Container**

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

```

503
504 Content-ID: <domainname.example.com> ----- | ebXML MIME
505 Content-Type: application/xml                ----- |
506
507 <Invoice>                                     ----- |
508   <Invoicedata>                               |
509     ...                                       |
510   </Invoicedata>                             |
511 </Invoice>                                     ----- |
    
```

Payload
Container

512 **7.5 Additional MIME Parameters**

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

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

518 **7.6 Reporting MIME Errors**

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

520 8 ebXML SOAP Extensions

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

- 524 • different software components are likely to be used to generate ebXML SOAP extension elements,
- 525 • an ebXML SOAP extension element is not always present or,
- 526 • the data contained in the ebXML SOAP extension element MAY be digitally signed separately from
527 the other ebXML SOAP extension elements.

528 8.1 XML Prolog

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

```
532 Content-Type: text/xml; charset="UTF-8"  
533 <?xml version="1.0" encoding="UTF-8"?>  
534  
535
```

536 8.1.1 XML Declaration

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

541 8.1.2 Encoding Declaration

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

545 If provided, the encoding declaration MUST NOT contain a value conflicting with the encoding used when
546 creating the SOAP *Message*. It is RECOMMENDED that UTF-8 be used when encoding the SOAP
547 *Message*.

548 If the character encoding cannot be determined by an XML processor using the rules specified in section
549 4.3.3 of [XML], the XML declaration and its contained encoding declaration SHALL be provided in the
550 ebXML SOAP **Header** Document.

551 Note: the encoding declaration is not required in an XML document according to XML v1.0 specification [XML].

552 8.2 ebXML SOAP Envelope extensions

553 In conformance with the [SOAP] specification, all extension element content MUST be namespace
554 qualified. All of the ebXML SOAP extension element content defined in this specification MUST be
555 namespace qualified to the ebXML SOAP **Envelope** extensions namespace as defined in section 8.2.1.

556 Namespace declarations (`xmlns` pseudo attribute) for the ebXML SOAP extensions MAY be included in
557 the SOAP **Envelope**, **Header** or **Body** elements, or directly in each of the ebXML SOAP extension
558 elements.

559 8.2.1 Namespace pseudo attribute

560 The namespace declaration for the ebXML SOAP **Envelope** extensions (`xmlns` pseudo attribute) (see
561 [XML Namespace]) has a REQUIRED value of "http://www.ebxml.org/namespaces/messageHeader".

562 **8.2.2 xsi:schemaLocation attribute**

563 The SOAP namespace:

564
565

```
http://schemas.xmlsoap.org/soap/envelope/
```

566 resolves to a schema that conforms to an early Working Draft version of the W3C XML Schema
567 specification, specifically identified by the following URI:

568
569

```
http://www.w3.org/1999/XMLSchema
```

570 The W3C XML Schema specification[XMLSchema] has since gone to Candidate Recommendation
571 status, effective October 24, 2000 and more recently to Proposed Recommendation effective March 30,
572 2001. Many, if not most, tool support for schema validation and validating XML parsers available at the
573 time that this specification was written have been designed to support the Candidate Recommendation
574 draft of the XML Schema specification[XMLSchema]. In addition, the ebXML SOAP extension element
575 schema has been defined using the Candidate Recommendation draft of the XML Schema
576 specification[XMLSchema] (see Appendix A).

577 In order to enable validating parsers and various schema validating tools to correctly process and parse
578 ebXML SOAP *Messages*, it has been necessary that the ebXML TR&P team adopt an equivalent, but
579 updated version of the SOAP schema that conforms to the W3C Candidate Recommendation draft of the
580 XML Schema specification[XMLSchema]. ebXML MSH implementations are strongly RECOMMENDED to
581 include the XMLSchema-instance namespace qualified **schemaLocation** attribute in the SOAP
582 **Envelope** element to indicate to validating parsers the location of the schema document that should be
583 used to validate the document. Failure to include the **schemaLocation** attribute will possibly preclude
584 *Receiving MSH* implementations from being able to validate messages received.

585 For example:

586
587
588
589
590

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsi="http://www.w3.org/2000/10/XMLSchema-instance"
  xsi:schemaLocation="http://schemas.xmlsoap.org/soap/envelope/
  http://ebxml.org/project_teams/transport/envelope.xsd" ...>
```

591 In addition, ebXML SOAP **Header** and **Body** extension element content must be similarly qualified so as
592 to identify the location that validating parsers can find the schema document that contains the ebXML
593 namespace qualified SOAP extension element definitions. Thus, the XMLSchema-instance namespace
594 qualified **schemaLocation** attribute should include a mapping of the ebXML SOAP **Envelope** extensions
595 namespace to its schema document in the same element that declares the ebXML SOAP **Envelope**
596 extensions namespace.

597 It is RECOMMENDED that use of a separate **schemaLocation** attribute be used so that tools that may
598 not correctly use the **schemaLocation** attribute to resolve schema for more than one namespace will still
599 be capable of validating an ebXML SOAP *message*. For example:

600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsi="http://www.w3.org/2000/10/XMLSchema-instance"
  xsi:schemaLocation="http://schemas.xmlsoap.org/soap/envelope/
  http://ebxml.org/project_teams/transport/envelope.xsd" ...>
  <SOAP-ENV:Header xmlns:eb="http://www.ebxml.org/namespaces/messageHeader"
    xsi:schemaLocation="http://www.ebxml.org/namespaces/messageHeader
    http://ebxml.org/project_teams/transport/messageHeaderV0_99.xsd" ...>
    <eb:MessageHeader ...> ...
  </eb:MessageHeader>
</SOAP-ENV:Header>
  <SOAP-ENV:Body xmlns:eb="http://www.ebxml.org/namespaces/messageHeader"
    xsi:schemaLocation="http://www.ebxml.org/namespaces/messageHeader
    http://ebxml.org/project_teams/transport/messageHeaderV0_99.xsd" ...>
    <eb:Manifest ...> ...
  </eb:Manifest>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

618 8.2.3 ebXML SOAP Extensions

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

- 620 • SOAP **Header** extensions:
 - 621 - **MessageHeader** – a REQUIRED element that contains routing information for the message
 - 622 (To/From, etc.) as well as other context information about the message.
 - 623 - **TraceHeaderList** – an element that contains entries that identifies the Message Service
 - 624 Handler(s) that sent and should receive the message. This element MAY be omitted.
 - 625 - **ErrorList** – an element that contains a list of the errors that are being reported against a previous
 - 626 message. The **ErrorList** element is only used if reporting an error on a previous message. This
 - 627 element MAY be omitted.
 - 628 - **Signature** – an element that contains a digital signature that conforms to [XMLDSIG] that signs
 - 629 data associated with the message. This element MAY be omitted.
 - 630 - **Acknowledgment**– an element that is used by a *Receiving MSH* to acknowledge to the *Sending*
 - 631 *MSH* that a previous message has been received. This element MAY be omitted.
 - 632 - **Via**– an element that is used to convey information to the next ebXML Message Service Handler
 - 633 that receives the message. This element MAY be omitted.
- 634 • SOAP **Body** extensions:
 - 635 - **Manifest** – an element that points to any data present either in the *Payload Container* or
 - 636 elsewhere, e.g. on the web. This element MAY be omitted.
 - 637 - **StatusRequest** – an element that is used to identify a message whose status is being requested.
 - 638 This element MAY be omitted.
 - 639 - **StatusResponse** – an element that is used by a MSH when responding to a request on the
 - 640 status of a message that was previously received. This element MAY be omitted.
 - 641 - **DeliveryReceipt** – an element used by the *To Party* that received a message, to let the *From*
 - 642 *Party* that sent the message know the message was received. This element MAY be omitted.

643 8.2.4 #wildcard element content

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

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

655 8.2.5 id attributes

656 Each of the ebXML SOAP extension elements listed above has an optional **id** attribute which is an XML
657 ID that MAY be added to provide for the ability to uniquely identify the element within the SOAP *Message*.
658 This MAY be used when applying a digital signature to the ebXML SOAP *Message* as individual ebXML
659 SOAP extension elements can be targeted for inclusion or exclusion by specifying a URI of "#<idvalue>"
660 in the **Reference** element.

661 8.3 SOAP Header element

662 The SOAP **Header** element is the first child element of the SOAP **Envelope** element. It MUST have a
663 namespace qualifier that matches the SOAP **Envelope** namespace declaration for the namespace
664 "http://schemas.xmlsoap.org/soap/envelope/". For example:
665

```

666 <SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/" ...>
667   <SOAP-ENV:Header>...</SOAP-ENV:Header>
668   <SOAP-ENV:Body>...</SOAP-ENV:Body>
669 </SOAP-ENV:Envelope>

```

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

672 8.4 MessageHeader element

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

675 The **MessageHeader** element is a composite element comprised of the following ten subordinate
676 elements:

- 677 • **From**
- 678 • **To**
- 679 • **CPAId**
- 680 • **ConversationId**
- 681 • **Service**
- 682 • **Action**
- 683 • **MessageData**
- 684 • **QualityOfServiceInfo**
- 685 • **SequenceNumber**
- 686 • **Description**

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

- 688 • SOAP **mustUnderstand**
- 689 • **Version**

690 In addition, the **MessageHeader** element MAY include an **id** attribute. See section 8.2.5 for details.

691 8.4.1 From and To elements

692 The REQUIRED **From** element identifies the *Party* that originated the message. The REQUIRED **To**
693 element identifies the *Party* that is the intended recipient of the message. Both **To** and **From** can contain
694 logical identifiers such as a DUNS number, or identifiers that also imply a physical location such as an
695 eMail address.

696 The **From** and the **To** elements each contain one or more **PartyId** child elements.

697 If either the **From** or **To** elements contain multiple **PartyId** elements, all members of the list must identify
698 the same organisation. Unless a single **type** value refers to multiple identification systems, a **type**
699 attribute value must not appear more than once in a single list of **PartyId** elements.

700 Note: This mechanism is particularly useful when transport of a message between the parties may involve multiple
701 intermediaries (see Sections 8.5.4, Multi-hop TraceHeader Sample and 10.3, ebXML Reliable Messaging Protocol).
702 More generally, the *From Party* should provide identification in all domains it knows in support of intermediaries
703 and destinations that may give preference to particular identification systems.

704 8.4.1.1 PartyID element

705 The **PartyId** element has a single attribute, **type** and content that is a string value. The **type** attribute
706 indicates the domain of names to which the string in the content of the **PartyId** element belongs. The
707 value of the **type** attribute MUST be mutually agreed and understood by each of the *Parties*. It is
708 RECOMMENDED that the value of the **type** attribute be a URI. It is further recommended that these
709 values be taken from the EDIRA (ISO 6523), EDIFACT ISO 9735 or ANSI ASC X12 I05 registries.

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

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

715
 716
 717
 718
 719
 720
 721
 722

```
<eb:From>
  <eb:PartyId eb:type="urn:duns">123456789</eb:PartyId>
  <eb:PartyId eb:type="SCAC">RDWY</PartyId>
</eb:From>
<eb:To>
  <eb:PartyId>mailto:joe@example.com</eb:PartyId>
</eb:To>
```

723 **8.4.2 CPAId element**

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

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

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

733

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

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

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

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

743 **8.4.3 ConversationId element**

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

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

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

753

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

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

757 **8.4.4 Service element**

758 The REQUIRED **Service** element identifies the *service* that acts on the message and it is specified by the
 759 designer of the *service*. The designer of the *service* may be:

- 760 • a standards organization, or
- 761 • an individual or enterprise

762 Note: In the context of an ebXML business process model, an *action* equates to the lowest possible role based
 763 activity in the [ebBPSS] (requesting or responding role) and a *service* is a set of related actions for an authorized
 764 role within a party.

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

766

```
<eb:Service>urn:services:SupplierOrderProcessing</eb:Service>
```

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

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

771 **8.4.4.1 type attribute**

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

775 If the **type** attribute is not present, the content of the **Service** element MUST be a URI [RFC2396]. If it is
 776 not a URI then report an error with an **errorCode** of **Inconsistent** and a **severity** of **Error** (see section
 777 11).

778 **8.4.5 Action element**

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

782

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

784 **8.4.6 MessageData element**

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

- 787 • **MessageId**
- 788 • **Timestamp**
- 789 • **RefToMessageId**
- 790 • **TimeToLive**

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

792

```
<eb:MessageData>
  <eb:MessageId>20001209-133003-28572@example.com</eb:MessageId>
  <eb:Timestamp>2001-02-15T11:12:12Z</eb:Timestamp>
  <eb:RefToMessageId>20001209-133003-28571@example.com</eb:RefToMessageId>
</eb:MessageData>
```

798 **8.4.6.1 MessageId element**

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

801 **8.4.6.2 Timestamp element**

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

804 **8.4.6.3 RefToMessageId element**

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

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

810 For Acknowledgment Messages, the **RefToMessageId** element is REQUIRED, and its value MUST be
811 the **MessageId** value of the ebXML Message being acknowledged. See also sections 8.13.4 and 10.

812 When **RefToMessageId** is contained inside either a **StatusRequest** or a **StatusResponse** element then
813 it identifies a Message whose current status is being queried (see section 9.1)

814 **8.4.6.4 TimeToLive element**

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

817 **8.4.7 QualityOfServiceInfo element**

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

- 820 • **deliverySemantics**
- 821 • **messageOrderSemantics**
- 822 • **deliveryReceiptRequested**

823 The **QualityOfServiceInfo** element SHALL be present if any of the attributes within the element need to
824 be set to their non-default value. The **deliverySemantics** attribute supports Reliable Messaging and is
825 discussed in detail in section 10. The **deliverySemantics** attribute indicates whether or not a message is
826 sent reliably.

827 **8.4.7.1 deliveryReceiptRequested attribute**

828 The **deliveryReceiptRequested** attribute is used by a *From Party* to indicate whether a message
829 received by the *To Party* should result in the *To Party* returning an acknowledgment message containing
830 a **DeliveryReceipt** element.

831 Note: To clarify the distinction between an acknowledgement message containing a **DeliveryReceipt** and a Reliable
832 Messaging Acknowledgement: (1) An acknowledgement message containing a **Delivery Receipt** indicates the *To*
833 *Party* has received the message. (2) The Reliable Messaging Acknowledgment indicates a MSH, possibly only an
834 intermediate MSH, has received the message.

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

837 Valid values for **deliveryReceiptRequested** are:

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

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

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

844 If the *To Party* can produce the Delivery Receipt of the type requested, then it MUST return to the *From Party* a message containing a **DeliveryReceipt** element.
845

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

848 If there are no errors in the message received and a **DeliveryReceipt** is being sent on its own, not as part
849 of message containing payload data, then the **Service** and **Action** MUST be set as follows:

- 850 • the **Service** element MUST be set to **uri:www.ebXML.org/messageService/**
- 851 • the **Action** element MUST be set to **DeliveryReceipt**

852 An example of **deliveryReceiptRequested** follows:

```
853
854 <eb:QualityOfServiceInfo eb:deliverySemantics="OnceAndOnlyOnce"
855     eb:messageOrderSemantics="Guaranteed"
856     eb:deliveryReceiptRequested="Unsigned" />
```

857 8.4.7.2 messageOrderSemantics attribute

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

- 860 • **Guaranteed** - The messages are passed to the receiving application in the order that the sending
861 application specified.
- 862 • **NotGuaranteed** - The messages may be passed to the receiving application in different order from
863 the order the sending application specified.

864 The default value for **messageOrderSemantics** is specified in the *CPA* or in **MessageHeader**. If a value
865 is not specified, the default value is **NotGuaranteed**.

866 If **messageOrderSemantics** is set to **Guaranteed**, the *To Party* MSH MUST correct invalid order of
867 messages using the value of **SequenceNumber** in the conversation specified by the **ConversationId**.
868 The **Guaranteed** semantics can be set only when **deliverySemantics** is **OnceAndOnlyOnce**. If
869 **messageOrderSemantics** is set to **Guaranteed** the **SequenceNumber** element MUST be present.

870 If **deliverySemantics** is not **OnceAndOnlyOnce** and **messageOrderSemantics** is set to **Guaranteed**
871 then report the error to the *From Party* with an **errorCode** of **Inconsistent** and a **severity** of **Error** (see
872 sections 10 and 11).

873 All messages sent within the same conversation, as identified by the **ConversationId** element, that have
874 a **deliverySemantics** attribute with a value of **OnceandOnlyOnce** SHALL each have the same value
875 **messageOrderSemantics** (either **Guaranteed** or **NotGuaranteed**).

876 If **messageOrderSemantics** is set to **NotGuaranteed**, then the *To Party* MSH does not need to correct
877 invalid order of messages.

878 If the *To Party* is unable to support the type of **messageOrderSemantics** requested, then the *To Party*
879 MUST report the error to the *From Party* using an **errorCode** of **NotSupported** and a **severity** of **Error**.
880 A sample of **messageOrderSemantics** follows.

```
881
882 <eb:QualityOfServiceInfo eb:deliverySemantics="OnceAndOnlyOnce"
883     eb:messageOrderSemantics="Guaranteed" />
```

884 8.4.8 SequenceNumber element

885 The **SequenceNumber** element indicates the sequence in which messages MUST be processed by a
886 *Receiving MSH*. The **SequenceNumber** is unique within the **ConversationId** and MSH. The *From Party*
887 MSH and the *To Party* MSH each set an independent **SequenceNumber** as the *Sending MSH* within the
888 **ConversationID**. It is set to zero on the first message from that MSH for a conversation and then
889 incremented by one for each subsequent message sent.

890 The **SequenceNumber** element MUST appear only when **deliverySemantics** has a value of
891 **OnceAndOnlyOnce** and **messageOrderSemantics** has a value of **Guaranteed**. If this criterion is not

892 met, an error MUST be reported to the From Party MSH with an **errorCode** of **Inconsistent** and a
 893 **severity** of **Error**.

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

898 If the implementation defined limit for saved out-of-sequence messages is reached, then the *Receiving*
 899 *MSH* MUST indicate a delivery failure to the *Sending MSH* with **errorCode** set to **DeliveryFailure** and
 900 **severity** set to **Error** (see section 11).

901 The **SequenceNumber** element is an integer value that is incremented by the *Sending MSH* (e.g. 0, 1, 2,
 902 3, 4...) for each application-prepared message sent by that MSH within the **ConversationId**. The next
 903 value of 99999999 in the increment is "0". The value of **SequenceNumber** consists of ASCII numerals in
 904 the range 0-99999999. In following cases, **SequenceNumber** takes the value "0":

- 905 1) First message from the *Sending MSH* within the conversation
- 906 2) First message after resetting **SequenceNumber** information by the *Sending MSH*
- 907 3) First message after wraparound (next value after 99999999)

908 The **SequenceNumber** element has a single attribute, **status**. This attribute is an enumeration, which
 909 SHALL have one of the following values:

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

912 When the **SequenceNumber** is set to "0" because of 1 or 2 above, the *Sending MSH* MUST set the
 913 **status** attribute of the message to **Reset**. In all other cases, including 3 above, the **status** attribute
 914 MUST be set to **Continue**.

915 A *Sending MSH* MUST wait before resetting the **SequenceNumber** of a conversation until it has received
 916 all of the *Acknowledgement Messages* for Messages previously sent for the conversation. Only when all
 917 the sent Messages are acknowledged, can the *Sending MSH* reset the **SequenceNumber**. An example
 918 of **SequenceNumber** follows.

919
 920

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

921 **8.4.9 Description element**

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

927 **8.4.10 version attribute**

928 The REQUIRED **version** attribute indicates the version of the *ebXML Message Service Header*
 929 Specification to which the ebXML SOAP **Header** extensions conform. Its purpose is to provide future
 930 versioning capabilities. The value of the **version** attribute MUST be "1.0". Future versions of this
 931 specification SHALL require other values of this attribute. The version attribute MUST be namespace
 932 qualified for the ebXML SOAP **Envelope** extensions namespace defined above.

933 **8.4.11 SOAP mustUnderstand attribute**

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

938 **8.4.12 MessageHeader Sample**

939 The following fragment demonstrates the structure of the *MessageHeader* element within the SOAP
 940 *Header*:

941
 942
 943
 944
 945
 946
 947
 948
 949
 950
 951
 952
 953
 954
 955
 956
 957
 958
 959

```
<eb:MessageHeader id="..." eb:version="1.0" SOAP-ENV:mustUnderstand="1">
  <eb:From><eb:PartyId>uri:example.com</eb:PartyId></eb:From>
  <eb:To eb:type="someType">
    <eb:PartyId eb:type="someType">QRS543</eb:PartyId>
  </eb:To>
  <eb:CPAId>http://www.ebxml.org/cpa/123456</eb:CPAId>
  <eb:ConversationId>987654321</eb:ConversationId>
  <eb:Service eb:type="myservicetypes">QuoteToCollect</eb:Service>
  <eb:Action>NewPurchaseOrder</eb:Action>
  <eb:MessageData>
    <eb:MessageId>mid:UUID-2</eb:MessageId>
    <eb:Timestamp>2000-07-25T12:19:05Z</eb:Timestamp>
    <eb:RefToMessageId>mid:UUID-1</eb:RefToMessageId>
  </eb:MessageData>
  <eb:QualityOfServiceInfo
    eb:deliverySemantics="OnceAndOnlyOnce"
    eb:deliveryReceiptRequested="Signed"/>
</eb:MessageHeader>
```

960 **8.5 TraceHeaderList element**

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

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

- 965 • the message is being sent over a single hop (see section 8.5.3), and
- 966 • the message is not being sent reliably (see section 10)

967 The *TraceHeaderList* element has three REQUIRED attributes as follows:

- 968 • SOAP *mustUnderstand* (See section 8.4.11 for details)
- 969 • SOAP *actor* attribute with the value "http://schemas.xmlsoap.org/soap/actor/next"
- 970 • *Version* (See section 8.4.10 for details)

971 In addition, the *TraceHeaderList* element MAY include an *id* attribute. See section 8.2.5 for details.

972 **8.5.1 SOAP actor attribute**

973 The *TraceHeaderList* element MUST contain a SOAP *actor* attribute with the value
 974 http://schemas.xmlsoap.org/soap/actor/next and be interpreted and processed as defined in the [SOAP]
 975 specification. This means that the *TraceHeaderList* element MUST be processed by the MSH that
 976 receives the message and SHOULD NOT be forwarded to the next MSH. A MSH that handles the
 977 *TraceHeaderList* element is REQUIRED to perform the function of appending a new *TraceHeader*
 978 element to the *TraceHeaderList* and (re)inserting it into the message for the next MSH.

979 **8.5.2 TraceHeader element**

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

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

- 985 • *Sender*
- 986 • *Receiver*
- 987 • *Timestamp*
- 988 • *#wildcard*

989 In addition, the *TraceHeader* element MAY include an *id* attribute. See section 8.2.5 for details.

990 **8.5.2.1 Sender element**

991 The *Sender* element is a composite element comprised of the following subordinate elements:

- 992 • *PartyId*
- 993 • *Location*

994 As with the *From* and *To* elements, multiple *PartyId* elements may be listed in the *Sender* element. This
995 allows receiving systems to resolve those identifiers to organizations using a preferred identification
996 scheme without prior agreement among all parties to a single scheme.

997 **8.5.2.1.1 PartyId element**

998 This element has the syntax and semantics described in Section 8.4.1.1, *PartyId* element. In this case,
999 the identified party is the sender of the message. This element may be used in a later message
1000 addressed to this party by including it in the *To* element of that message.

1001 **8.5.2.1.2 Location element**

1002 This element contains the URL of the Sender's Message Service Handler. Unless there is another URL
1003 identified within the *CPA* or in *MessageHeader* (section 8.4.2), the recipient of the message uses the
1004 URL to send a message, when required that:

- 1005 • responds to an earlier message
- 1006 • acknowledges an earlier message
- 1007 • reports an error in an earlier message.

1008 **8.5.2.2 Receiver element**

1009 The *Receiver* element is a composite element comprised of the following subordinate elements:

- 1010 • *PartyId*
- 1011 • *Location*

1012 As with the *From* and *To* elements, multiple *PartyId* elements may be listed in the *Receiver* element.
1013 This allows sending systems to resolve those identifiers to organisations using a preferred identification
1014 scheme without prior agreement among all parties to a single scheme.

1015 The descendant elements of the *Receiver* element (*PartyId* and *Location*) are implemented in the same
1016 manner as the Sender element (see sections 8.5.2.1.1 and 8.5.2.1.2).

1017 **8.5.2.3 Timestamp element**

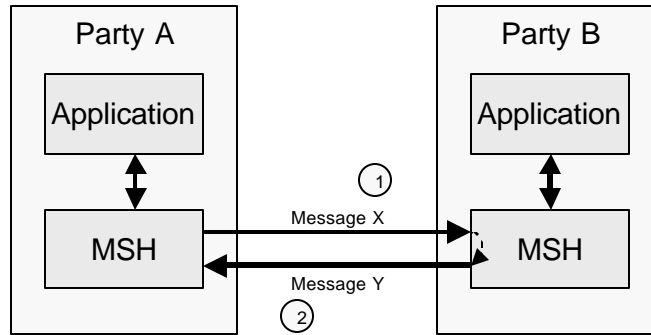
1018 The *Timestamp* element is the time the individual *TraceHeader* was created. It is in the same format as
1019 in the *Timestamp* element in the *MessageData* element (section 8.4.6.2).

1020 **8.5.2.4 #wildcard element**

1021 Refer to section 8.2.4 for discussion of #wildcard element handling.

1022 **8.5.3 Single Hop TraceHeader Sample**

1023 A single hop message is illustrated by the diagram below.



1024

1025 **Figure 8-1 Single Hop Message**

1026 The content of the corresponding messages could include:

- 1027 • Transmission 1 - Message X From Party A To Party B

1028

1029

1030

1031

1032

1033

1034

1035

1036

1037

1038

1039

1040

1041

1042

1043

1044

1045

1046

1047

1048

1049

1050

1051

1052

1053

1054

1055

1056

1057

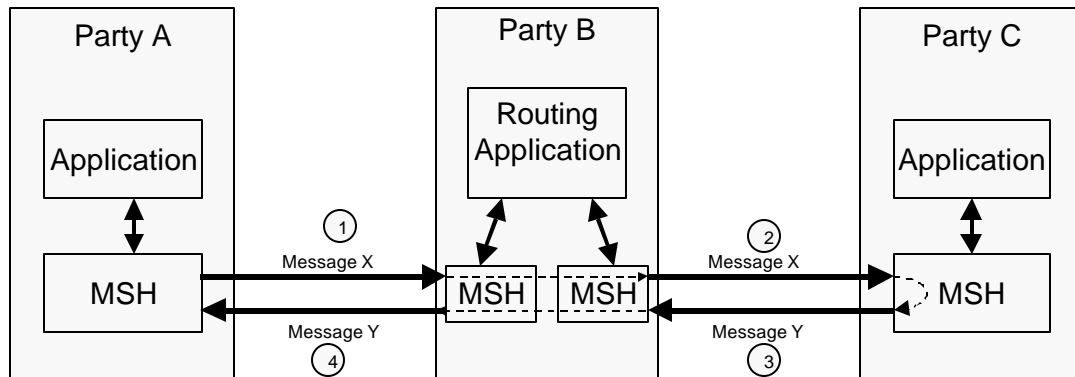
```

<eb:MessageHeader eb:id="..." eb:version="1.0" SOAP-ENV:mustUnderstand="1">
  <eb:From>
    <eb:PartyId>urn:myscheme.com:id:PartyA-id</eb:PartyId>
  </eb:From>
  <eb:To>
    <eb:PartyId>urn:myscheme.com:id:PartyB-id</eb:PartyId>
  </eb:To>
  <eb:ConversationId>219cdj89dj2398djfjn</eb:ConversationId>
  ...
  <eb:MessageData>
    <eb:MessageId>29dmridj103kvna</eb:MessageId>
    ...
  </eb:MessageData>
  ...
</eb:MessageHeader>

<eb:TraceHeaderList eb:id="..." eb:version="1.0" SOAP-ENV:mustUnderstand="1">
  <eb:TraceHeader>
    <eb:Sender>
      <eb:PartyId>urn:myscheme.com:id:PartyA-id</eb:PartyId>
      <eb:Location>http://PartyA.com/PartyAMsh</eb:Location>
    </eb:Sender>
    <eb:Receiver>
      <eb:PartyId>urn:myscheme.com:id:PartyB-id</eb:PartyId>
      <eb:Location>http://PartyB.com/PartyBMsh</eb:Location>
    </eb:Receiver>
    <eb:Timestamp>2000-12-16T21:19:35Z</eb:Timestamp>
  </eb:TraceHeader>
</eb:TraceHeaderList>
    
```

1058 **8.5.4 Multi-hop TraceHeader Sample**

1059 Multi-hop messages are not sent directly from one party to another, instead they are sent via an
 1060 intermediate party, as illustrated by the diagram below:



1061

1062 **Figure 8-2 Multi-hop Message**

1063 The content of the corresponding messages could include:

- 1064 • Transmission 1 - Message X From Party A To Party B

1065

```

1066 <eb:MessageHeader eb:id="..." eb:version="1.0" SOAP-ENV:mustUnderstand="1">
1067   <eb:From>
1068     <eb:PartyId>urn:myscheme.com:id:PartyA-id</eb:PartyId>
1069   </eb:From>
1070   <eb:To>
1071     <eb:PartyId>urn:myscheme.com:id:PartyC-id</eb:PartyId>
1072   </eb:To>
1073   <eb:ConversationId>219cdj89dj2398djfjn</eb:ConversationId>
1074   ...
1075   <eb:MessageData>
1076     <eb:MessageId>29dmridj103kvna</eb:MessageId>
1077     ...
1078   </eb:MessageData>
1079   ...
1080 </eb:MessageHeader>
1081
1082 <eb:TraceHeaderList eb:id="..." eb:version="1.0" SOAP-ENV:mustUnderstand="1"
1083   SOAP-ENV:actor="http://schemas.xmlsoap.org/soap/actor/next">
1084   <eb:TraceHeader>
1085     <eb:Sender>
1086       <eb:PartyId>urn:myscheme.com:id:PartyA-id</eb:PartyId>
1087       <eb:Location>http://PartyA.com/PartyAMsh</eb:Location>
1088     </eb:Sender>
1089     <eb:Receiver>
1090       <eb:Location>http://PartyB.com/PartyBMSH</eb:Location>
1091     </eb:Receiver>
1092     <eb:Timestamp>2000-12-16T21:19:35Z</eb:Timestamp>
1093   </eb:TraceHeader>
1094 </eb:TraceHeaderList>
    
```

- 1095 • Transmission 2 - Message X From Party B To Party C

```

1097 <eb:MessageHeader eb:id="..." eb:version="1.0" SOAP-ENV:mustUnderstand="1">
1098   <eb:From>
1099     <eb:PartyId>urn:myscheme.com:id:PartyA-id</eb:PartyId>
1100   </eb:From>
1101   <eb:To>
1102     <eb:PartyId>urn:myscheme.com:id:PartyC-id</eb:PartyId>
1103   </eb:To>
1104   <eb:ConversationId>219cdj89dj2398djfjn</eb:ConversationId>
1105   ...
1106   <eb:MessageData>
    
```

1107
1108
1109
1110
1111
1112
1113
1114
1115
1116
1117
1118
1119
1120
1121
1122
1123
1124
1125
1126
1127
1128
1129
1130
1131
1132
1133
1134
1135
1136
1137

```

    <eb:MessageId>29dmridj103kvna</eb:MessageId>
    ...
  </eb:MessageData>
  ...
</eb:MessageHeader>
<eb:TraceHeaderList eb:id="..." eb:version="1.0" SOAP-ENV:mustUnderstand="1"
  SOAP-ENV:actor="http://schemas.xmlsoap.org/soap/actor/next">
  <eb:TraceHeader>
    <eb:Sender>
      <eb:PartyId>urn:myscheme.com:id:PartyA-id</eb:PartyId>
      <eb:Location>http://PartyA.com/PartyAMsh</eb:Location>
    </eb:Sender>
    <eb:Receiver>
      <eb:PartyId>urn:myscheme.com:id:PartyB-id</eb:PartyId>
      <eb:Location>http://PartyB.com/PartyBMsh</eb:Location>
    </eb:Receiver>
    <eb:Timestamp>2000-12-16T21:19:35Z</eb:Timestamp>
  </eb:TraceHeader>
  <eb:TraceHeader>
    <eb:Sender>
      <eb:PartyId>urn:myscheme.com:id:PartyB-id</eb:PartyId>
      <eb:Location>http://PartyB.com/PartyAMsh</eb:Location>
    </eb:Sender>
    <eb:Receiver>
      <eb:PartyId>urn:myscheme.com:id:PartyC-id</eb:PartyId>
      <eb:Location>http://PartyC.com/PartyBMsh</eb:Location>
    </eb:Receiver>
    <eb:Timestamp>2000-12-16T21:19:45Z</eb:Timestamp>
  </eb:TraceHeader>
</eb:TraceHeaderList>

```

1138 **8.6 Acknowledgment Element**

1139 The **Acknowledgment** element is an optional element that is used by one Message Service Handler to
1140 indicate that another Message Service Handler has received a message. The **RefToMessageId** in a
1141 message containing an **Acknowledgement** element is used to identify the message being acknowledged
1142 by its **MessageId**.

1143 The **Acknowledgment** element consists of the following elements and attributes:

- 1144 • a **Timestamp** element
- 1145 • a **From** element
- 1146 • zero or more **ds:Reference** element(s)
- 1147 • a REQUIRED SOAP **mustUnderstand** attribute (See section 8.4.11 for details)
- 1148 • a REQUIRED SOAP **actor** attribute
- 1149 • a REQUIRED **version** attribute (See section 8.4.10 for details)
- 1150 • an **id** attribute (See section 8.2.5 for details)

1151 **8.6.1 Timestamp element**

1152 The **Timestamp** element is a value representing the time that the message being acknowledged was
1153 received by the **Party** generating the acknowledgment message. It must conform to an [XMLSchema]
1154 **timeInstant** (section 8.4.6.2).

1155 **8.6.2 From element**

1156 This is the same element as the **From** element within **MessageHeader** element (see section 8.4.1).
1157 However, when used in the context of an **Acknowledgment** element, it contains the identifier of the **Party**
1158 that is generating the **acknowledgment message**.

1159 If the **From** element is omitted then the **Party** that is sending the element is identified by the **From**
1160 element in the **MessageHeader** element.

1161 **8.6.3 ds:Reference element**

1162 An Acknowledgment MAY be used to enable non-repudiation of receipt by a MSH by including one or
 1163 more **Reference** elements from the [XMLDSIG] namespace (<http://www.w3.org/2000/09/xmlsig#>) taken,
 1164 or derived, from the message being acknowledged. The **Reference** element(s) MUST be namespace
 1165 qualified to the aforementioned namespace and MUST conform to the XML Signature[XMLDSIG]
 1166 specification.

1167 **8.6.4 SOAP actor attribute**

1168 The **Acknowledgment** element MUST contain a SOAP **actor** attribute with the value
 1169 <http://schemas.xmlsoap.org/soap/actor/next> and be interpreted and processed as defined in the [SOAP]
 1170 specification. This means that the **Acknowledgment** element MUST be processed by the MSH that
 1171 receives the message and SHOULD NOT be forwarded to the next MSH.

1172 **8.6.5 Acknowledgement Sample**

1173 An example of the **Acknowledgement** element is given below:

1174
 1175
 1176
 1177
 1178
 1179
 1180
 1181

```
<eb:Acknowledgment SOAP-ENV:mustUnderstand="1" eb:version="1.0"
  SOAP-ENV:actor="http://schemas.xmlsoap.org/soap/actor/next">
  <eb:Timestamp>2001-03-09T12:22:30Z</eb:Timestamp>
  <eb:From>
    <eb:PartyId>uri:www.example.com</eb:PartyId>
  </eb:From>
</eb:Acknowledgment>
```

1182 **8.7 Via element**

1183 The **Via** element is an ebXML extension to the SOAP **Header** that is used to convey information to the
 1184 next ebXML Message Service Handler (MSH) that receives the message.

1185 Note: this MSH can be a MSH operated by an intermediary or by the *To Party*. In particular, the **Via** element is used
 1186 to hold data that can vary from one hop to another.

1187 The **Via** element MUST contain the following attributes:

- 1188 • **id** attribute (See section 8.2.5)
- 1189 • **version** attribute (See section 8.4.10 for details)
- 1190 • SOAP **MustUnderstand** attribute
- 1191 • SOAP **actor** attribute

1192 The **Via** element MUST also contain one or more of the following elements or attributes:

- 1193 • **syncReply** attribute
- 1194 • **reliableMessagingMethod** attribute
- 1195 • **ackRequested** attribute
- 1196 • **CPAId** element

1197 The **Via** element MAY also contain the following elements:

- 1198 • **Service** element
- 1199 • **Action** element

1200 **8.7.1 SOAP mustUnderstand attribute**

1201 The REQUIRED SOAP **mustUnderstand** attribute, namespace qualified to the SOAP **Envelope**
 1202 namespace (<http://schemas.xmlsoap.org/soap/envelope/>), indicates that the contents of the **Via** element
 1203 MUST be understood by a receiving process or else the message MUST be rejected in accordance with
 1204 [SOAP]. This attribute MUST have a value of '1' (true). In accordance with the [SOAP] specification, a
 1205 receiving *ebXML Message Service* implementation that does not provide support for the **Via** element
 1206 MUST respond with a SOAP **Fault** with a **faultCode** of **MustUnderstand**.

1207 8.7.2 SOAP actor attribute

1208 The **Via** element MUST contain a SOAP **actor** attribute with the value
1209 <http://schemas.xmlsoap.org/soap/actor/next> and be interpreted and processed as defined in the [SOAP]
1210 specification. This means that the **Via** element MUST be processed by the MSH that receives the
1211 message and SHOULD NOT be forwarded to the next MSH.

1212 8.7.3 syncReply attribute

1213 The **syncReply** attribute is used only if the data communication protocol is *synchronous* (e.g. HTTP). It is
1214 an [XMLSchema] boolean. If the communication protocol is not *synchronous*, then the value of
1215 **syncReply** is ignored. If the **syncReply** attribute is not present, it is semantically equivalent to its
1216 presence with a value of "false". If the **syncReply** attribute is present with a value of **true**, the MSH must
1217 return the response from the application or business process in the payload of the *synchronous* reply
1218 message. See also the description of **syncReply** in the [ebCPP] specification.

1219 8.7.4 reliableMessagingMethod attribute

1220 The **reliableMessagingMethod** attribute is an enumeration that SHALL have one of the following values:

- 1221 • **ebXML**
- 1222 • **Transport**

1223 The default implied value for this attribute is **ebXML**.

1224 8.7.5 ackRequested attribute

1225 The **ackRequested** attribute is an enumeration that SHALL have one of the following values:

- 1226 • **Signed**
- 1227 • **Unsigned**
- 1228 • **None**

1229 The default implied value for this attribute is **None**. This attribute is used to indicate to the *Receiving MSH*
1230 whether an acknowledgment message is expected, and if so, whether the acknowledgment message
1231 should be signed by the *Receiving MSH*. Refer to section 10.2.5 for a complete discussion as to the use
1232 of this attribute.

1233 8.7.6 CPAId element

1234 The **CPAId** element is a string that identifies the parameters that govern the exchange of messages
1235 between two MSH instances. It has the same meaning as the **CPAId** in the **MessageHeader** except that
1236 the parameters identified by the **CPAId** apply just to the exchange of messages between the two MSH
1237 instances rather than between the *Parties* identified in the **To** and **From** elements of the **MessageHeader**
1238 (section 8.4.2). This allows different parameters, transport protocols, etc, to be used on different hops
1239 when a message is passed through intermediaries.

1240 If the **CPAId** element is present, the identified parameter values SHOULD be used instead of the values
1241 identified by the **CPAId** in the **MessageHeader** element.

1242 8.7.7 Service and Action elements

1243 The **Service** and **Action** elements have the same meaning as the **Service** and **Action** elements in the
1244 **MessageHeader** element (see sections 8.4.4 and 8.4.5) except that they are interpreted and acted on by
1245 the next MSH whether or not the MSH is operated by the *To Party*.

1246 The designer of the service or business process that is using the *ebXML Message Service* defines the
1247 values used for **Service** and **Action**.

1248 The **Service** and **Action** elements are OPTIONAL. However, if the **Service** element is present then the
1249 **Action** element MUST also be present and vice versa.

1250 **8.7.8 Via element Sample**

1251 The following is a sample *Via* element.

1252
1253
1254
1255
1256
1257
1258
1259

```
<eb:Via SOAP-ENV:mustUnderstand="1" eb:version="1.0"
  SOAP-ENV:actor="http://schemas.xmlsoap.org/soap/actor/next"
  eb:syncReply="false">
  <eb:CPAId>yaddaydda</eb:CPAId>
  <eb:Service>urn:services:Proxy</eb:Service>
  <eb:Action>LogActivity</eb:Action>
</eb:Via>
```

1260 **8.8 ErrorList element**

1261 The existence of an *ErrorList* element within the SOAP *Header* element indicates that the message that
1262 is identified by the *RefToMessageId* in the *MessageHeader* element has an error.

1263 The *ErrorList* element consists of one or more *Error* elements and the following attributes:

- 1264 • *id* attribute
- 1265 • SOAP *mustUnderstand* attribute (See section 8.4.11 for details)
- 1266 • *version* attribute (See section 8.4.10 for details)
- 1267 • *highestSeverity* attribute

1268 If there are no errors to be reported then the *ErrorList* element MUST NOT be present.

1269 **8.8.1 id attribute**

1270 The *id* attribute uniquely identifies the *ErrorList* element within the document (See section 8.2.5).

1271 **8.8.2 highestSeverity attribute**

1272 The *highestSeverity* attribute contains the highest severity of any of the *Error* elements. Specifically, if
1273 any of the *Error* elements have a *severity* of *Error* then *highestSeverity* must be set to *Error*, otherwise
1274 set *highestSeverity* to *Warning*.

1275 **8.8.3 Error element**

1276 An *Error* element consists of the following attributes:

- 1277 • *codeContext*
- 1278 • *errorCode*
- 1279 • *severity*
- 1280 • *location*
- 1281 • *xml:lang*
- 1282 • *id* (See section 8.2.5 for details)

1283 The content of the *Error* element contains an error message.

1284 **8.8.3.1 codeContext attribute**

1285 The REQUIRED *codeContext* attribute identifies the namespace or scheme for the *errorCodes*. It
1286 MUST be a URI. Its default value is <http://www.ebxml.org/messageServiceErrors>. If it does not have
1287 the default value, then it indicates that an implementation of this specification has used its own
1288 *errorCodes*.

1289 Use of non-ebXML values for *errorCodes* is NOT RECOMMENDED. In addition, an implementation of
1290 this specification MUST NOT use its own *errorCodes* if an existing *errorCode* as defined in this section
1291 has the same or very similar meaning.

1292 **8.8.3.2 errorCode attribute**

1293 The REQUIRED **errorCode** attribute indicates the nature of the error in the message in error. Valid
 1294 values for the **errorCode** and a description of the code's meaning are given in sections 8.8.5.1 and
 1295 8.8.5.2

1296 **8.8.3.3 severity attribute**

1297 The REQUIRED **severity** attribute indicates the severity of the error. Valid values are:
 1298 • **Warning** - This indicates that although there is an error, other messages in the conversation will still
 1299 be generated in the normal way.
 1300 • **Error** - This indicates that there is an unrecoverable error in the message and no further messages
 1301 will be generated as part of the conversation.

1302 **8.8.3.4 location attribute**

1303 The **location** attribute points to the part of the message that is in error.
 1304 If an error exists in an ebXML element and the element is "well formed" (see [XML]), then the content of
 1305 the **location** attribute MUST be an [XPointer].
 1306 If the error is associated with the MIME envelope that wraps the SOAP envelope and the ebXML
 1307 Payload, then **location** contains the `content-id` of the MIME part that is in error, in the format
 1308 `cid:23912480wsr`, where the text after the ":" is the value of the MIME part's `content-id`.

1309 **8.8.3.5 Error element Content**

1310 The content of the error message provides a narrative description of the error in the language defined by
 1311 the **xml:lang** attribute. Typically, it will be the message generated by the XML parser or other software
 1312 that is validating the message. This means that the content is defined by the vendor/developer of the
 1313 software that generated the **Error** element.
 1314 The **xml:lang** attribute must comply with the rules for identifying languages specified in [XML].
 1315 The content of the **Error** element can be empty.

1316 **8.8.4 ErrorList Sample**

1317 An example of an **ErrorList** element is given below.

1318
 1319
 1320
 1321
 1322
 1323
 1324
 1325

```
<eb:ErrorList eb:id='3490sdo9', eb:highestSeverity="error" eb:version="1.0"
    SOAP-ENV:mustUnderstand="1">
  <eb:Error eb:errorCode='SecurityFailure' eb:severity="Error"
    eb:location='URI_of_ds:Signature_goes_here' xml:lang="us-en">
    Validation of signature failed </eb:Error>
  <eb:Error ...> ... </eb:Error>
</eb:ErrorList>
```

1326 **8.8.5 errorCode values**

1327 This section describes the values for the **errorCode** element (see section 8.8.3.2) used in a *message*
 1328 *reporting an error*. They are described in a table with three headings:
 1329 • the first column contains the value to be used as an **errorCode**, e.g. **SecurityFailure**
 1330 • the second column contains a "Short Description" of the **errorCode**.
 1331 Note: this narrative MUST NOT be used in the content of the **Error** element.
 1332 • the third column contains a "Long Description" that provides an explanation of the meaning of the
 1333 error and provides guidance on when the particular **errorCode** should be used.

1334 **8.8.5.1 Reporting Errors in the ebXML Elements**

1335 The following list contains error codes that can be associated with ebXML elements:

1336

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 is consistent with the rules and constraints contained in this specification, but 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.

1337 **8.8.5.2 Non-XML Document Errors**

1338 The following are error codes that identify errors not associated with the ebXML elements:

1339

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.

1340 **8.9 ds:Signature element**

1341 An ebXML Message may be digitally signed to provide security countermeasures. Zero or more
1342 **ds:Signature** elements, belonging to the [XMLDSIG] defined namespace MAY be present in the SOAP

1343 **Header.** The **ds:Signature** element MUST be namespace qualified in accordance with [XMLDSIG]. The
 1344 structure and content of the **ds:Signature** element MUST conform to the [XMLDSIG] specification. If
 1345 there is more than one **ds:Signature** element contained within the SOAP **Header**, the first MUST
 1346 represent the digital signature of the ebXML Message as signed by the *From Party* MSH in conformance
 1347 with section 12. Additional **ds:Signature** elements MAY be present, but their purpose is undefined by
 1348 this specification.

1349 Refer to section 12 for a detailed discussion on how to construct the **ds:Signature** element when digitally
 1350 signing an ebXML Message.

1351 8.10 SOAP Body Extensions

1352 The SOAP **Body** element is the second child element of the SOAP **Envelope** element. It MUST have a
 1353 namespace qualifier that matches the SOAP **Envelope** namespace declaration for the namespace
 1354 "http://schemas.xmlsoap.org/soap/envelope/". For example:

```
1355 <SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/" ...>
1356   <SOAP-ENV:Header>...</SOAP-ENV:Header>
1357   <SOAP-ENV:Body>...</SOAP-ENV:Body>
1358 </SOAP-ENV:Envelope>
```

1360 The SOAP **Body** element contains the ebXML SOAP **Body** extension element content as follows:

- 1361 • **Manifest** element
- 1362 • **StatusRequest** element
- 1363 • **StatusResponse** element
- 1364 • **DeliveryReceipt** element

1365 Each is defined in the following sections.

1366 8.11 Manifest element

1367 The **Manifest** element is a composite element consisting of one or more **Reference** elements. Each
 1368 **Reference** element identifies data associated with the message, whether included as part of the
 1369 message as payload document(s) contained in a *Payload Container*, or remote resources accessible via
 1370 a URL. It is RECOMMENDED that no payload data be present in the SOAP **Body**. The purpose of the
 1371 **Manifest** is as follows:

- 1372 • to make it easier to directly extract a particular payload associated with this ebXML Message,
- 1373 • to allow an application to determine whether it can process the payload without having to parse it.

1374 The **Manifest** element is comprised of the following attributes and elements, each of which is described
 1375 below:

- 1376 • an **id** attribute
- 1377 • a REQUIRED **version** attribute (See section 8.4.10 for details)
- 1378 • one or more **Reference** elements
- 1379 • **#wildcard**

1380 8.11.1 id attribute

1381 The **Manifest** element MUST have an **id** attribute that is an XML ID (See section 8.2.5).

1382 8.11.2 #wildcard element

1383 Refer to section 8.2.4 for discussion of #wildcard element handling.

1384 8.11.3 Reference element

1385 The **Reference** element is a composite element consisting of the following subordinate elements:

- 1386 • **Schema** - information about the schema(s) that define the instance document identified in the parent
 1387 **Reference** element

- 1388 • **Description** - a textual description of the payload object referenced by the parent **Reference** element
- 1389 • **#wildcard** - any namespace-qualified element content belonging to a foreign namespace

1390 The **Reference** element itself is an [XLINK] simple link. XLINK is presently a Candidate Recommendation
 1391 (CR) of the W3C. It should be noted that the use of XLINK in this context is chosen solely for the purpose
 1392 of providing a concise vocabulary for describing an association. Use of an XLINK processor or engine is
 1393 NOT REQUIRED, but MAY prove useful in certain implementations.

1394 The **Reference** element has the following attribute content in addition to the element content described
 1395 above:

- 1396 • **id** - an XML ID for the **Reference** element,
- 1397 • **xlink:type** - this attribute defines the element as being an XLINK simple link. It has a fixed value of
 1398 'simple',
- 1399 • **xlink:href** - this REQUIRED attribute has a value that is the URI of the payload object referenced. It
 1400 SHALL conform to the [XLINK] specification criteria for a simple link.
- 1401 • **xlink:role** - this attribute identifies some resource that describes the payload object or its purpose. If
 1402 present, then it SHALL have a value that is a valid URI in accordance with the [XLINK] specification,
- 1403 • Any other namespace-qualified attribute MAY be present. A *Receiving MSH* MAY choose to ignore
 1404 any foreign namespace attributes other than those defined above.

1405 **8.11.3.1 Schema element**

1406 If the item being referenced has schema(s) of some kind that describe it (e.g. an XML Schema, DTD, or a
 1407 database schema), then the **Schema** element SHOULD be present as a child of the **Reference** element.
 1408 It provides a means of identifying the schema and its version defining the payload object identified by the
 1409 parent **Reference** element. The **Schema** element contains the following attributes:

- 1410 • **location** - the REQUIRED URI of the schema
- 1411 • **version** – a version identifier of the schema

1412 **8.11.3.2 Description element**

1413 The **Reference** element MAY contain zero or more **Description** elements. The **Description** is a textual
 1414 description of the payload object referenced by the parent **Reference** element. The language of the
 1415 description is defined by a REQUIRED **xml:lang** attribute. The **xml:lang** attribute MUST comply with the
 1416 rules for identifying languages specified in [XML]. This element is provided to allow a human readable
 1417 description of the payload object identified by the parent **Reference** element. If multiple **Description**
 1418 elements are present, each SHOULD have a unique **xml:lang** attribute value. An example of a
 1419 **Description** element follows.

1420
 1421

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

1422 **8.11.3.3 #wildcard element**

1423 Refer to section 8.2.4 for discussion of #wildcard element handling.

1424 **8.11.4 References included in a Manifest**

1425 The designer of the business process or information exchange that is using ebXML Messaging decides
 1426 what payload data is referenced by the **Manifest** and the values to be used for **xlink:role**.

1427 **8.11.5 Manifest Validation**

1428 If an **xlink:href** attribute contains a URI that is a content id (URI scheme "cid") then a MIME part with
 1429 that `content-id` MUST be present in the *Payload Container* of the message. If it is not, then the error
 1430 SHALL be reported to the *From Party* with an **errorCode** of **MimeProblem** and a **severity** of **Error**.

1431 If an **xlink:href** attribute contains a URI that is not a content id (URI scheme "cid"), and that URI cannot
 1432 be resolved, then it is an implementation decision on whether to report the error. If the error is to be

1433 reported, then it SHALL be reported to the *From Party* with an **errorCode** of **MimeProblem** and a
 1434 **severity** of **Error**.

1435 **8.11.6 Manifest Sample**

1436 The following fragment demonstrates a typical **Manifest** for a message with a single payload MIME body
 1437 part:

```
1438
1439 <eb:Manifest eb:id="Manifest" eb:version="1.0">
1440   <eb:Reference eb:id="pay01"
1441     xlink:href="cid:payload-1"
1442     xlink:role="http://regrep.org/gci/purchaseOrder">
1443     <eb:Schema eb:location="http://regrep.org/gci/purchaseOrder/po.xsd" eb:version="1.0"/>
1444     <eb:Description xml:lang="en-us">Purchase Order for 100,000 widgets</eb:Description>
1445   </eb:Reference>
1446 </eb:Manifest>
```

1447 **8.12 StatusRequest Element**

1448 The **StatusRequest** element is an immediate child of a SOAP **Body** and is used to identify an earlier
 1449 message whose status is being requested (see section 9.1).

1450 The **StatusRequest** element consists of the following elements and attributes:

- 1451 • a REQUIRED **RefToMessageId** element
- 1452 • a REQUIRED **version** attribute (See section 8.4.10 for details)
- 1453 • an **id** attribute (See section 8.2.5 for details)

1454 **8.12.1 StatusRequest Sample**

1455 An example of the **StatusRequest** element is given below:

```
1456
1457 <eb:StatusRequest eb:version="1.0" >
1458   <eb:RefToMessageId>323210:e52151ec74:-7ffc@xtacy</eb:RefToMessageId>
1459 </eb:StatusRequest>
```

1460 **8.13 StatusResponse element**

1461 The **StatusResponse** element is used by one MSH to respond to a request on the status of the
 1462 processing of a message that was previously sent (see also section 9.1).

1463 The **StatusResponse** element consists of the following elements and attributes:

- 1464 • a REQUIRED **RefToMessageId** element
- 1465 • a **Timestamp** element
- 1466 • a REQUIRED **version** attribute (See section 8.4.10 for details)
- 1467 • a **messageStatus** attribute
- 1468 • an **id** attribute (See section 8.2.5 for details)

1469 **8.13.1 RefToMessageId element**

1470 A REQUIRED **RefToMessageId** element that contains the **MessageId** of the message whose status is
 1471 being reported.

1472 **8.13.2 Timestamp element**

1473 The **Timestamp** element contains the time that the message, whose status is being reported, was
 1474 received (section 8.4.6.2.). This MUST be omitted if the message whose status is being reported is
 1475 **NotRecognized** or the request was **Unauthorized**.

1476 **8.13.3 messageStatus attribute**

1477 The **messageStatus** attribute identifies the status of the message that is identified by the
 1478 **RefToMessageId** element. It SHALL be set to one of the following values:

- 1479 • **Unauthorized** – the Message Status Request is not authorized or accepted
- 1480 • **NotRecognized** – the message identified by the **RefToMessageId** element in the **StatusResponse**
- 1481 element is not recognized
- 1482 • **Received** – the message identified by the **RefToMessageId** element in the **StatusResponse**
- 1483 element has been received by the MSH

1484 Note: if a Message Status Request is sent after the elapsed time indicated by *persistDuration* has passed since the

1485 message being queried was sent, then the Message Status Response may indicate that the **MessageId** was

1486 **NotRecognized** as the **MessageId** is no longer in persistent storage.

1487 8.13.4 StatusResponse Sample

1488 An example of the **StatusResponse** element is given below:

1489
1490
1491
1492
1493

```
<eb:StatusResponse eb:version="1.0" eb:messageStatus="Received">
  <eb:RefToMessageId>323210:e52151ec74:-7ffc@xtacy</eb:RefToMessageId>
  <eb:Timestamp>2001-03-09T12:22:30Z</eb:Timestamp>
</eb:StatusResponse>
```

1494 8.14 DeliveryReceipt element

1495 The **DeliveryReceipt** element is an optional element that is used by the *To Party* that received a

1496 message, to let the *From Party* that sent the original message, know that the message was received. The

1497 **RefToMessageId** in a message containing a **DeliveryReceipt** element is used to identify the message

1498 being for which the receipt is being generated by its **MessageId**.

1499 The **DeliveryReceipt** element consists of the following elements and attributes:

- 1500 • an **id** attribute (See section 8.2.5)
- 1501 • a REQUIRED **version** attribute (See section 8.4.10 for details)
- 1502 • a **Timestamp** element
- 1503 • zero or more **ds:Reference** element(s)

1504 8.14.1 Timestamp element

1505 The **Timestamp** element is a value representing the time that the message for which a **DeliveryReceipt**

1506 element is being generated was received by the *To Party*. It must conform to an [XMLSchema]

1507 **dateTime** element.

1508 8.14.2 ds:Reference element

1509 An Acknowledgment MAY be used to enable non-repudiation of receipt by a MSH by including one or

1510 more **Reference** elements from the [XMLDSIG] namespace (<http://www.w3.org/2000/09/xmldsig#>) taken,

1511 or derived, from the message being acknowledged. The **Reference** element(s) MUST be namespace

1512 qualified to the aforementioned namespace and MUST conform to the XML Signature [XMLDSIG]

1513 specification.

1514 8.14.3 DeliveryReceipt Sample

1515 An example of the **DeliveryReceipt** element is given below:

1516
1517
1518
1519
1520
1521
1522
1523

```
<eb:DeliveryReceipt eb:version="1.0">
  <eb:Timestamp>2001-03-09T12:22:30Z</eb:Timestamp>
  <ds:Reference URI="cid://blahblahblah/">
    <ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#dsa-sha1"/>
    <ds:DigestValue>...</ds:DigestValue>
  </ds:Reference>
</eb:DeliveryReceipt>
```

1524 8.15 Combining ebXML SOAP Extension Elements

1525 This section describes how the various ebXML SOAP extension elements may be used in combination.

1526 8.15.1 Manifest element

1527 The *Manifest* element MUST be present if there is any data associated with the message that is not
1528 present in the *Header Container*. This applies specifically to data in the *Payload Container* or elsewhere,
1529 e.g. on the web.

1530 8.15.2 MessageHeader element

1531 The *MessageHeader* element MUST be present in every message.

1532 8.15.3 TraceHeaderList element

1533 The *TraceHeaderList* element MAY be present in any message. It MUST be present if the message is
1534 being sent reliably (see section 10) or over multiple hops (see section 8.5.4).

1535 8.15.4 StatusRequest element

1536 A *StatusRequest* element MUST NOT be present with the following elements:

- 1537 • a *Manifest* element
- 1538 • an *ErrorList* element

1539 8.15.5 StatusResponse element

1540 This element MUST NOT be present with the following elements:

- 1541 • a *Manifest* element
- 1542 • a *StatusRequest* element
- 1543 • an *ErrorList* element with a *highestSeverity* attribute set to *Error*

1544 8.15.6 ErrorList element

1545 If the *highestSeverity* attribute on the *ErrorList* is set to *Warning*, then this element MAY be present
1546 with any other element.

1547 If the *highestSeverity* attribute on the *ErrorList* is set to *Error*, then this element MUST NOT be present
1548 with the following:

- 1549 • a *Manifest* element
- 1550 • a *StatusResponse* element

1551 8.15.7 Acknowledgment element

1552 An *Acknowledgment* element MAY be present on any message.

1553 8.15.8 Delivery Receipt element

1554 A *DeliveryReceipt* element may be present on any message.

1555 8.15.9 Signature element

1556 One or more *ds:Signature* elements MAY be present on any message.

1557 8.15.10 Via element

1558 One-and-only-one *Via* element MAY be present in any message.

1559 9 Message Service Handler Services

1560 The Message Service Handler MAY support two services that are designed to help provide smooth
1561 operation of a Message Handling Service implementation:

- 1562 • Message Status Request
- 1563 • Message Service Handler Ping

1564 If a *Receiving MSH* does not support the service requested, it SHOULD return a SOAP fault with a
1565 **faultCode** of **MustUnderstand**. Each service is described below.

1566 9.1 Message Status Request Service

1567 The Message Status Request Service consists of the following:

- 1568 • A Message Status Request message containing details regarding a message previously sent is sent
1569 to a Message Service Handler (MSH)
- 1570 • The Message Service Handler receiving the request responds with a Message Status Response
1571 message.

1572 A Message Service Handler SHOULD respond to Message Status Requests for messages that have
1573 been sent reliably (see section 10) and the **MessageId** in the **RefToMessageId** is present in *persistent*
1574 *storage* (see section 10.1.1).

1575 A Message Service Handler MAY respond to Message Status Requests for messages that have not been
1576 sent reliably.

1577 A Message Service SHOULD NOT use the Message Status Request Service to implement Reliable
1578 Messaging.

1579 9.1.1 Message Status Request Message

1580 A Message Status Request message consists of an *ebXML Message* containing no *ebXML Payload* and
1581 the following elements in the SOAP **Header**:

- 1582 • a **MessageHeader** element
- 1583 • a **TraceHeaderList** element
- 1584 • a **StatusRequest** element
- 1585 • a **ds:Signature** element

1586 The **TraceHeaderList** and the **ds:Signature** elements MAY be omitted (see sections 8.5 and 8.15.8).

1587 The **MessageHeader** element MUST contain the following:

- 1588 • a **From** element that identifies the *Party* that created the message status request message
- 1589 • a **To** element identifying a *Party* who should receive the message. If a **TraceHeader** was present on
1590 the message whose status is being checked, this MUST be set using the **Receiver** of the message.
1591 All **PartyId** elements present in the **Receiver** element SHOULD be included in this **To** element.
- 1592 • a **Service** element that contains: **uri:www.ebxml.org/messageService/**
- 1593 • an **Action** element that contains **StatusRequest**

1594 The message is then sent to the *To Party*.

1595 The **RefToMessageId** element in **StatusRequest** element in the SOAP **Body** contains the **MessageId** of
1596 the message whose status is being queried.

1597 9.1.2 Message Status Response Message

1598 Once the *To Party* receives the Message Status Request message, they SHOULD generate a Message
1599 Status Response message consisting of no ebXML Payload and the following elements in the SOAP
1600 **Header** and **Body**.

- 1601 • a **MessageHeader** element
- 1602 • a **TraceHeaderList** element
- 1603 • an **Acknowledgment** element
- 1604 • a **StatusResponse** element (see section 8.13)
- 1605 • a **ds:Signature** element

1606 The **TraceHeaderList**, **Acknowledgment** and **ds:Signature** elements MAY be omitted (see sections
1607 8.5, 8.15.7 and 8.15.8).

1608 The **MessageHeader** element MUST contain the following:

- 1609 • a **From** element that identifies the sender of the Message Status Response message
- 1610 • a **To** element that is set to the value of the **From** element in the Message Status Request message
- 1611 • a **Service** element that contains the value: **uri:www.ebxml.org/messageService/**
- 1612 • an **Action** element that contains **StatusResponse**
- 1613 • a **RefToMessageId** that identifies the Message Status Request message.

1614 The message is then sent to the *To Party*.

1615 9.1.3 Security Considerations

1616 Parties who receive a Message Status Request message SHOULD always respond to the message.
1617 However, they MAY ignore the message instead of responding with **messageStatus** set to
1618 **Unauthorized** if they consider that the sender of the message is unauthorized. The decision process
1619 that results in this course of action is implementation dependent.

1620 9.2 Message Service Handler Ping Service

1621 The Message Service Handler Ping Service enables one MSH to determine if another MSH is operating.
1622 It consists of:

- 1623 • sending a Message Service Handler Ping message to a MSH, and
- 1624 • the MSH that receives the Ping responding with a Message Service Handler Pong message.

1625 9.2.1 Message Service Handler Ping Message

1626 A Message Service Handler Ping (MSH Ping) message consists of an *ebXML Message* containing no
1627 ebXML Payload and the following elements in the SOAP **Header**:

- 1628 • a **MessageHeader** element
- 1629 • a **TraceHeaderList** element
- 1630 • a **ds:Signature** element

1631 The **TraceHeaderList** and the **ds:Signature** elements MAY be omitted (see sections 8.5 and 8.15.8).

1632 The **MessageHeader** element MUST contain the following:

- 1633 • a **From** element that identifies the *Party* creating the MSH Ping message
- 1634 • a **To** element that identifies the *Party* that is being sent the MSH Ping message
- 1635 • a **CPAId** element
- 1636 • a **ConversationId** element
- 1637 • a **Service** element that contains: **uri:www.ebxml.org/messageService/**
- 1638 • an **Action** element that contains **Ping**

1639 The message is then sent to the *To Party*.

1640 9.2.2 Message Service Handler Pong Message

1641 Once the *To Party* receives the MSH Ping message, they MAY generate a Message Service Handler
1642 Pong (MSH Pong) message consisting of an ebXML Message containing no ebXML Payload and the
1643 following elements in the SOAP **Header**:

- 1644 • a **MessageHeader** element
- 1645 • a **TraceHeaderList** element
- 1646 • an **Acknowledgment** element
- 1647 • an OPTIONAL **ds:Signature** element

1648 The **TraceHeaderList**, **Acknowledgment** and **ds:Signature** elements MAY be omitted (see sections
1649 8.5, 8.15.7 and 8.15.8).

1650 The **MessageHeader** element MUST contain the following:

- 1651 • a **From** element that identifies the creator of the MSH Pong message
- 1652 • a **To** element that identifies a *Party* that generated the MSH Ping message
- 1653 • a **CPAId** element
- 1654 • a **ConversationId** element
- 1655 • a **Service** element that contains the value: **uri:www.ebxml.org/messageService/**
- 1656 • an **Action** element that contains the value **Pong**
- 1657 • a **RefToMessageId** that identifies the MSH Ping message.

1658 The message is then sent to the *To Party*.

1659 **9.2.3 Security Considerations**

1660 Parties who receive a MSH Ping message SHOULD always respond to the message. However, there is
1661 a risk that some parties might use the MSH Ping message to determine the existence of a Message
1662 Service Handler as part of a security attack on that MSH. Therefore, recipients of a MSH Ping MAY
1663 ignore the message if they consider that the sender of the message received is unauthorized or part of
1664 some attack. The decision process that results in this course of action is implementation dependent.

1665 10 Reliable Messaging

1666 Reliable Messaging defines an interoperable protocol such that the two Message Service Handlers
1667 (MSH) can “reliably” exchange messages that are sent using “reliable messaging” semantics, resulting in
1668 the *To Party* receiving the message once and only once.

1669 Reliability is achieved by a *Receiving MSH* responding to a message with an *Acknowledgment Message*.

1670 10.1.1 Persistent Storage and System Failure

1671 A MSH that supports Reliable Messaging MUST keep messages that are sent or received reliably in
1672 *persistent storage*. In this context *persistent storage* is a method of storing data that does not lose
1673 information after a system failure or interruption.

1674 This specification recognizes that different degrees of resilience may be realized depending on the
1675 technology that is used to persist the data. However, as a minimum, persistent storage that has the
1676 resilience characteristics of a hard disk (or equivalent) SHOULD be used. It is strongly RECOMMENDED
1677 though that implementers of this specification use technology that is resilient to the failure of any single
1678 hardware or software component.

1679 After a system interruption or failure, a MSH MUST ensure that messages in persistent storage are
1680 processed in the same way as if the system failure or interruption had not occurred. How this is done is
1681 an implementation decision.

1682 In order to support the filtering of duplicate messages, a *Receiving MSH* SHOULD save the **MessageId**
1683 in *persistent storage*. It is also RECOMMENDED that the following be kept in *Persistent Storage*:

- 1684 • the complete message, at least until the information in the message has been passed to the
1685 application or other process that needs to process it
- 1686 • the time the message was received, so that the information can be used to generate the response to
1687 a Message Status Request (see section 9.1)
- 1688 • complete response message

1689 10.1.2 Methods of Implementing Reliable Messaging

1690 Support for Reliable Messaging MAY be implemented in one of the following two ways:

- 1691 • using the ebXML Reliable Messaging protocol, or
- 1692 • using ebXML SOAP structures together with commercial software products that are designed to
1693 provide reliable delivery of messages using alternative protocols.

1694 10.2 Reliable Messaging Parameters

1695 This section describes the parameters required to control reliable messaging. This parameter information
1696 can be specified in the *CPA* or in the **MessageHeader** (section 8.4.2).

1697 10.2.1 Delivery Semantics

1698 The **deliverySemantics** value MUST be used by the *From Party* MSH to indicate whether the Message
1699 MUST be sent reliably. Valid values are:

- 1700 • **OnceAndOnlyOnce** - The message must be sent using a **reliableMessagingMethod** that will result
1701 in the application or other process at the *To Party* receiving the message once and only once
- 1702 • **BestEffort** - The reliable delivery semantics are not used. In this case, the value of
1703 **reliableMessagingMethod** is ignored.

1704 The value for **deliverySemantics** is specified in the *CPA* or in **MessageHeader** (section 8.4.2). The
1705 default value for **deliverySemantics** is **BestEffort**.

1706 If **deliverySemantics** is set to **OnceAndOnlyOnce**, the *From Party* MSH and the *To Party* MSH must
1707 adopt a reliable messaging behavior that describes how messages are resent in the case of failure. The
1708 **deliverySemantic** value of **OnceAndOnlyOnce** will cause duplicate messages to be ignored.

1709 If **deliverySemantics** is set to **BestEffort**, a MSH that received a message that it is unable to deliver
1710 MUST NOT take any action to recover or otherwise notify anyone of the problem. The MSH that sent the
1711 message MUST NOT attempt to recover from any failure. This means that duplicate messages might be
1712 delivered to an application and persistent storage of messages is not required.

1713 If the *To Party* is unable to support the type of delivery semantics requested, the *To Party* SHOULD
1714 report the error to the *From Party* using an **ErrorCode** of **NotSupported** and a **Severity** of **Error**.

1715 **10.2.2 mshTimeAccuracy**

1716 The **mshTimeAccuracy** parameter indicates the minimum accuracy a *Receiving MSH* keeps the clocks it
1717 uses when checking, for example, **TimeToLive**. Its value is in the format "mm:ss" which indicates the
1718 accuracy in minutes and seconds.

1719 **10.2.3 TimeToLive**

1720 The **TimeToLive** value indicates the time by which a message should be delivered to and processed by
1721 the *To Party*. It must conform to an XML Schema `timeInstant`.

1722 In this context, the **TimeToLive** has expired if the time of the internal clock of the *Receiving MSH* is
1723 greater than the value of **TimeToLive** for the message.

1724 When setting a value for **TimeToLive** it is RECOMMENDED that the *From Party's* MSH takes into
1725 account the accuracy of its own internal clocks as well as the **mshTimeAccuracy** parameter for the
1726 *Receiving MSH* indicating the accuracy to which a MSH will keep its internal clocks. How a MSH ensures
1727 that its internal clocks are kept sufficiently accurate is an implementation decision.

1728 If the *To Party's* MSH receives a message where **TimeToLive** has expired, it SHALL send a message to
1729 the *From Party* MSH, reporting that the **TimeToLive** of the message has expired. This message SHALL
1730 be comprised of an **ErrorList** containing an error that has the **errorCode** attribute set to
1731 **TimeToLiveExpired**, and the **severity** attribute set to **Error**.

1732 **10.2.4 reliableMessagingMethod**

1733 The **reliableMessagingMethod** attribute SHALL have one of the following values:

- 1734 • **ebXML**
- 1735 • **Transport**

1736 The default implied value for this attribute is **ebXML** and is case sensitive. Refer to section 8.7.4 for
1737 discussion of the use of this attribute.

1738 **10.2.5 ackRequested**

1739 The **ackRequested** value is used by the *Sending MSH* to request that the *Receiving MSH* returns an
1740 *acknowledgment message* with an **Acknowledgment** element.

1741 Valid values for **ackRequested** are:

- 1742 • **Unsigned** - requests that an unsigned Acknowledgement is requested
- 1743 • **Signed** - requests that a signed Acknowledgement is requested, or
- 1744 • **None** - indicates that no Acknowledgement is requested.

1745 The default value is **None**.

1746 **10.2.6 retries**

1747 The **retries** value is an integer value that specifies the maximum number of times a *Sending MSH*
1748 SHOULD attempt to redeliver an unacknowledged *message* using the same Communications Protocol.

1749 **10.2.7 retryInterval**

1750 The *retryInterval* value is a time value, expressed as a duration in accordance with the [XMLSchema]
 1751 timeDuration data type. This value specifies the minimum time the *Sending MSH* MUST wait between
 1752 retries, if an *Acknowledgment Message* is not received.

1753 **10.2.8 persistDuration**

1754 The *persistDuration* value is the minimum length of time, expressed as a [XMLSchema] timeDuration,
 1755 that data from a reliably sent *Message*, is kept in *Persistent Storage* by a *Receiving MSH*.

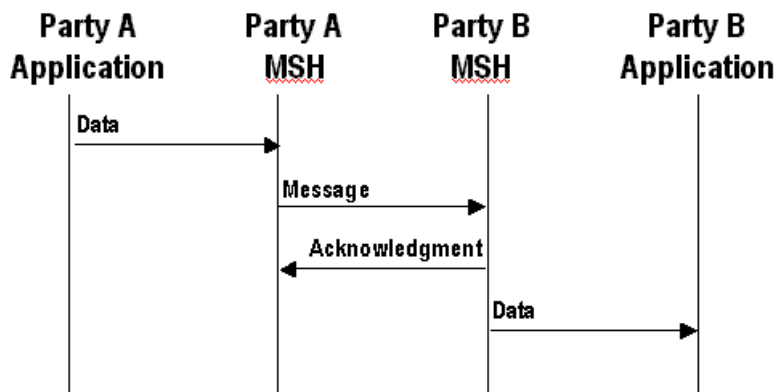
1756 If the *persistDuration* has passed since the message was first sent, a *Sending MSH* SHOULD NOT
 1757 resend a message with the same *MessageId*.

1758 If a message cannot be sent successfully before *persistDuration* has passed, then the *Sending MSH*
 1759 should report a delivery failure (see section 10.4).

1760 **10.3 ebXML Reliable Messaging Protocol**

1761 The ebXML Reliable Messaging Protocol described in this section MUST be followed if the
 1762 *deliverySemantics* parameter/element is set to *OnceAndOnlyOnce* and the *reliableMessagingMethod*
 1763 parameter/element is set to *ebXML* (the default).

1764 The ebXML Reliable Messaging Protocol is illustrated by the figure below.



1765
 1766 **Figure 10-1 Indicating that a message has been received**

1767 The receipt of the *Acknowledgment Message* indicates that the message being acknowledged has been
 1768 successfully received and either processed or persisted by the *Receiving MSH*.

1769 An *Acknowledgment Message* MUST contain a *MessageData* element with a *RefToMessageId* that
 1770 contains the same value as the *MessageId* element in the *message being acknowledged*.

1771 **10.3.1 Sending Message Behavior**

1772 If a MSH is given data by an application that needs to be sent reliably (i.e. the *deliverySemantics* is set
 1773 to *OnceAndOnlyOnce*), then the MSH MUST do the following:

- 1774 1. Create a message from components received from the application that includes a *TraceHeader*
 1775 element identifying the sender and the receiver as described in Section 8.5.2 *TraceHeader* element.
- 1776 2. Save the message in *persistent storage* (see section 10.1.1)
- 1777 3. Send the message to the *Receiver MSH*
- 1778 4. Wait for the *Receiver MSH* to return an *Acknowledgment Message* and, if it does not or a transient
 1779 error is returned, then take the appropriate action as described in section 10.3.4

1780 **10.3.2 Receiving Message Behavior**

- 1781 If the **deliverySemantics** for the received message is set to **OnceAndOnlyOnce** then do the following:
- 1782 1. If the message is just an acknowledgement (i.e. the **Service** element is set to
 1783 <http://www.ebxml.org/namespaces/messageService/MessageAcknowledgment> and **Action** is set to
 1784 **Acknowledgment**), then:
- 1785 a) Look for a message in *persistent storage* that has a **MessageId** that is the same as the value of
 1786 **RefToMessageId** on the received Message
- 1787 b) If a message is found in *persistent storage* then mark the persisted message as delivered
- 1788 2. Otherwise, if the message is not just an acknowledgement, then check to see if the message is a
 1789 duplicate (e.g. there is a **MessageId** held in *persistent storage* that was received earlier that
 1790 contains the same value for the **MessageId**)
- 1791 c) If the message is not a duplicate then do the following:
- 1792 i) Save the **MessageId** of the received message in *persistent storage*. As an implementation
 1793 decision, the whole message MAY be stored if there are other reasons for doing so.
- 1794 ii) If the received message contains a **RefToMessageId** element then do the following:
- 1795 (1) Look for a message in *persistent storage* that has a **MessageId** that is the same as the
 1796 value of **RefToMessageId** on the received Message
- 1797 (2) If a message is found in *persistent storage* then mark the persisted message as delivered
- 1798 iii) Generate an *Acknowledgement Message* in response (see section 10.3.3).
- 1799 d) If the message is a duplicate, then do the following:
- 1800 i) Look in persistent storage for the first response to the received message and resend it (i.e. it
 1801 contains a **RefToMessageId** that matches the **MessageId** of the received message)
- 1802 ii) If a message was found in *persistent storage* then resend the persisted message back to the
 1803 MSH that sent the received message,
- 1804 iii) If no message was found in *persistent storage*, then:
- 1805 (1) if **syncReply** is set to **True** and if the CPA indicates an application response is included,
 1806 ignore the received message (i.e. no message was generated in response to the
 1807 message, or the processing of the earlier message is not yet complete)
- 1808 (2) if **syncReply** is set to **False** then generate an *Acknowledgement Message* (see section
 1809 10.3.3).

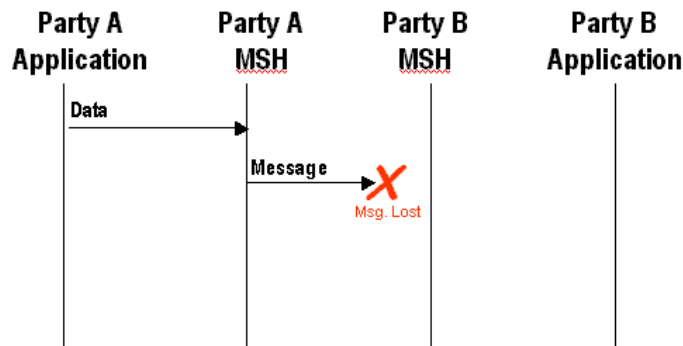
1810 **10.3.3 Generating an Acknowledgement Message**

- 1811 An *Acknowledgement Message* MUST be generated whenever a message is received with:
- 1812 • **deliverySemantics** set to **OnceAndOnlyOnce** and
 - 1813 • **reliableMessagingMethod** set to **ebXML** (the default).
- 1814 As a minimum, it MUST contain a **MessageData** element with a **RefToMessageId** that contains the same
 1815 value as the **MessageId** element in the *message being acknowledged*.
- 1816 If **ackRequested** in the **Via** of the received message is set to **Signed** or **Unsigned** then the
 1817 acknowledgement message MUST also contain an **Acknowledgement** element.
- 1818 Depending on the value of the **syncReply** parameter, the *Acknowledgement Message* can also be sent
 1819 at the same time as the response to the received message. In this case, the values for the
 1820 **MessageHeader** elements of the *Acknowledgement Message* are set by the designer of the Service.
- 1821 If an **Acknowledgment** element is being sent on its own, then the value of the **MessageHeader**
 1822 elements MUST be set as follows:

- 1823 • The Service element MUST be set to: uri:www.ebxml.org/messageService/
- 1824 • The Action element MUST be set to Acknowledgment.
- 1825 • The From element MAY be populated with the To element extracted from the message received, or it
- 1826 MAY be set using the Receiver from the last TraceHeader in the message that has just been
- 1827 received. In either case, all PartyId elements from the message received SHOULD be included in this
- 1828 From element.
- 1829 • The To element MAY be populated with the From element extracted from the message received, or it
- 1830 MAY be set using the Sender from the last TraceHeader in the message that has just been received.
- 1831 In either case, all PartyId elements from the message received SHOULD be included in this To
- 1832 element.
- 1833 • The RefToMessageId element MUST be set to the MessageId of the message that has just been
- 1834 received

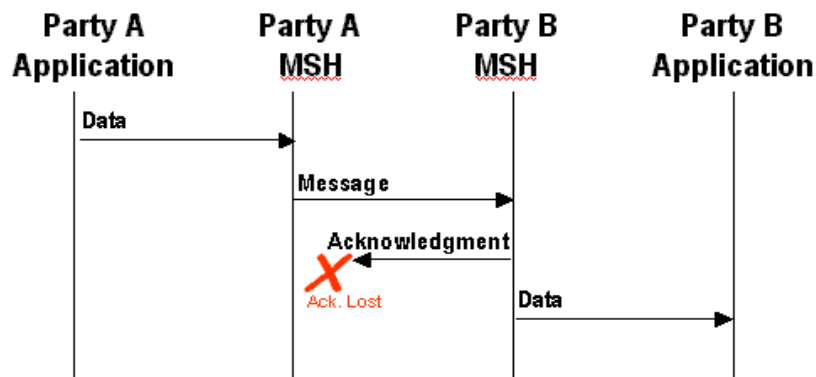
1835 **10.3.4 Resending Lost Messages and Duplicate Filtering**

1836 This section describes the behavior that is required by the sender and receiver of a message in order to
 1837 handle when messages are lost. A message is "lost" when a *Sending MSH* does not receive a response
 1838 to a message. For example, it is possible that a *message* was lost, for example:



1839
 1840 **Figure 10-2 Undelivered Message**

1841 It is also possible that the *Acknowledgment Message* was lost, for example:



1842
 1843 **Figure 10-3 Lost Acknowledgment Message**

1844 The rules that apply are as follows:

- 1845 • The *Sending MSH* MUST resend the original message if an *Acknowledgment Message* has not been
- 1846 received from the *Receiving MSH* and the following are both true:

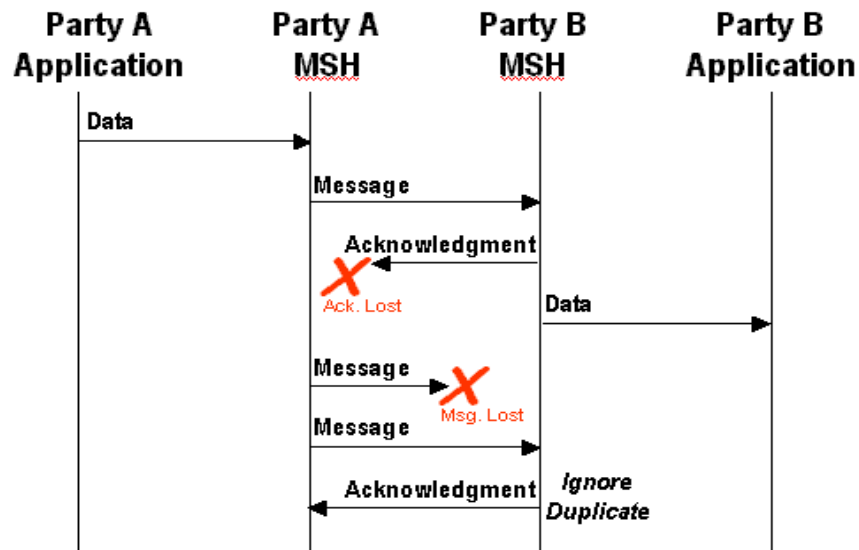
- 1847 a) At least the time specified in the **retryInterval** has passed since the message was last sent, and
- 1848 b) The message has been resent less than the number of times specified in the **retries** Parameter
- 1849 • If the *Sending MSH* does not receive an *Acknowledgment Message* after the maximum number of
- 1850 retries, the *Sending MSH* SHOULD notify the application and/or system administrator function of the
- 1851 failure to receive an acknowledgement.
- 1852 • If the *Sending MSH* detects an unrecoverable communications protocol error at the transport protocol
- 1853 level, the *Sending MSH* SHOULD resend the message.

1854 **10.3.5 Duplicate Message Handling**

1855 In the context of this specification, a duplicate message is:

- 1856 • an “identical message” is a *message* that contains, apart from an additional **TraceHeader** element,
- 1857 the same ebXML SOAP **Header, Body** and ebXML *Payload* as the earlier *message* that was sent.
- 1858 • a “duplicate message” is a *message* that contains the same **MessageId** as an earlier message that
- 1859 was received.
- 1860 • the “first message” is the message with the earliest **Timestamp** in the **MessageData** element that
- 1861 has the same **RefToMessageId** as the duplicate message.

1862



1863

1864 **Figure 10-4 Resending Unacknowledged Messages**

1865 The diagram above shows the behavior that MUST be followed by the sending and *Receiving MSH* that
 1866 are sent with **deliverySemantics** of **OnceAndOnlyOnce**. Specifically:

- 1867 1) The sender of the *message* (e.g. Party A) MUST resend the “identical message” if no
- 1868 *Acknowledgment Message* is received.
- 1869 2) When the recipient (Party B) of the *message* receives a “duplicate message”, it MUST resend to the
- 1870 sender (Party A) a message identical to the *first message* that was sent to the sender Party A).
- 1871 3) The recipient of the *message* (Party B) MUST NOT forward the message a second time to the
- 1872 application/process.

1873 **10.4 Failed Message Delivery**

1874 If a message sent with **deliverySemantics** set to **OnceAndOnlyOnce** cannot be delivered, the MSH or
1875 process SHOULD send a delivery failure notification to the *From Party*. The delivery failure notification
1876 message contains:

- 1877 • a **From** element that identifies the *Party* who detected the problem
- 1878 • a **To** element that identifies the *From Party* that created the message that could not be delivered
- 1879 • a **Service** element and **Action** element set as described in 11.5
- 1880 • an **Error** element with a severity of:
 - 1881 - **Error** if the party who detected the problem could not transmit the message (e.g. the
 - 1882 communications transport was not available)
 - 1883 - **Warning** if the message was transmitted, but an *acknowledgment message* was not received.
 - 1884 This means the message probably was not delivered although there is a small probability it was.
- 1885 • an **ErrorCode** of **DeliveryFailure**

1886 It is possible that an error message with an **Error** element with an **ErrorCode** set to **DeliveryFailure**
1887 cannot be delivered successfully for some reason. If this occurs, then the *From Party* that is the ultimate
1888 destination for the error message SHOULD be informed of the problem by other means. How this is done
1889 is outside the scope of this specification.

1890 11 Error Reporting and Handling

1891 This section describes how one ebXML Message Service Handler (MSH) reports errors it detects in an
1892 ebXML Message to another MSH. The *ebXML Message Service* error reporting and handling is to be
1893 considered as a layer of processing above the SOAP processor layer. This means the ebXML MSH is
1894 essentially an application-level handler of a *SOAP Message* from the perspective of the SOAP Processor.
1895 The SOAP processor MAY generate SOAP **Fault** messages if it is unable to process the message. A
1896 *Sending MSH* MUST be prepared to accept and process these SOAP **Faults**.

1897 It is possible for the ebXML MSH software to cause a SOAP fault to be generated and returned to the
1898 sender of a *SOAP Message*. In this event, the returned message MUST conform to the [SOAP]
1899 specification processing guidelines for SOAP **Faults**.

1900 An ebXML *SOAP Message* that reports an error that has a **highestSeverity** of **Warning** SHALL NOT be
1901 reported or returned as a SOAP **Fault**.

1902 11.1 Definitions

1903 For clarity, two phrases are defined that are used in this section:

- 1904 • “message in error” - A *message* that contains or causes an error of some kind
- 1905 • “message reporting the error” - A *message* that contains an ebXML **ErrorList** element that describes
1906 the error(s) found in a message in error.

1907 11.2 Types of Errors

1908 One MSH needs to report to another MSH errors in a message in error. For example, errors associated
1909 with:

- 1910 • ebXML namespace qualified content of the *SOAP Message* document (see section 8)
- 1911 • reliable messaging failures (see section 10)
- 1912 • security (see section 12)

1913 Unless specified to the contrary, all references to “an error” in the remainder of this specification imply
1914 any or all of the types of errors listed above.

1915 Errors associated with Data Communication protocols are detected and reported using the standard
1916 mechanisms supported by that data communication protocol and do not use the error reporting
1917 mechanism described here.

1918 11.3 When to generate Error Messages

1919 When a MSH detects an error in a message it is strongly RECOMMENDED that the error is reported to
1920 the MSH that sent the message that had an error if:

- 1921 • the Error Reporting Location (see section 11.4) to which the message reporting the error should be
1922 sent can be determined, and
- 1923 • the message in error does not have an **ErrorList** element with **highestSeverity** set to **Error**.

1924 If the Error Reporting Location cannot be found or the message in error has an **ErrorList** element with
1925 **highestSeverity** set to **Error**, it is RECOMMENDED that:

- 1926 • the error is logged, and
- 1927 • the problem is resolved by other means, and
- 1928 • no further action is taken.

1929 **11.3.1 Security Considerations**

1930 Parties that receive a Message containing an error in the header SHOULD always respond to the
1931 message. However, they MAY ignore the message and not respond if they consider that the message
1932 received is unauthorized or is part of some security attack. The decision process resulting in this course
1933 of action is implementation dependent.

1934 **11.4 Identifying the Error Reporting Location**

1935 The Error Reporting Location is a URI that is specified by the sender of the message in error that
1936 indicates where to send a *message reporting the error*.

1937 The **ErrorURI** implied by the *CPA*, identified by the **CPAId** on the message, SHOULD be used. If no
1938 **ErrorURI** is implied by the *CPA* and a **TraceHeaderList** is present in the message in error, the value of
1939 the **Location** element in the **Sender** of the topmost **TraceHeader** MUST be used. Otherwise, the
1940 recipient MAY resolve an **ErrorURI** using the **From** element of the message in error. If this is not
1941 possible, no error will be reported to the sending *Party*.

1942 Even if the message in error cannot be successfully analyzed or parsed, MSH implementers SHOULD try
1943 to determine the Error Reporting Location by other means. How this is done is an implementation
1944 decision.

1945 **11.5 Service and Action Element Values**

1946 An **ErrorList** element can be included in a SOAP **Header** that is part of a *message* being sent as a result
1947 of processing of an earlier message. In this case, the values for the **Service** and **Action** elements are
1948 set by the designer of the *Service*.

1949 An **ErrorList** element can also be included in an SOAP **Header** that is not being sent as a result of the
1950 processing of an earlier message. In this case, if the **highestSeverity** is set to **Error**, the values of the
1951 **Service** and **Action** elements MUST be set as follows:

- 1952 • The **Service** element MUST be set to: *uri:www.ebxml.org/messageService/*
- 1953 • The **Action** element MUST be set to **MessageError**.

1954 If the **highestSeverity** is set to **Warning**, the **Service** and **Action** elements MUST NOT be used.

1955 12 Security

1956 The *ebXML Message Service*, by its very nature, presents certain security risks. A Message Service may
1957 be at risk by means of:

- 1958 • Unauthorized access
- 1959 • Data integrity and/or confidentiality attacks (e.g. through man-in-the-middle attacks)
- 1960 • Denial-of-Service and spoofing

1961 Each security risk is described in detail in the ebXML Technical Architecture Security Specification
1962 [ebTASEC].

1963 Each of these security risks MAY be addressed in whole, or in part, by the application of one, or a
1964 combination, of the countermeasures described in this section. This specification describes a set of
1965 profiles, or combinations of selected countermeasures, selected to address key risks based upon
1966 commonly available technologies. Each of the specified profiles includes a description of the risks that
1967 are not addressed.

1968 Application of countermeasures SHOULD be balanced against an assessment of the inherent risks and
1969 the value of the asset(s) that might be placed at risk.

1970 12.1 Security and Management

1971 No technology, regardless of how advanced it might be, is an adequate substitute to the effective
1972 application of security management policies and practices.

1973 It is strongly RECOMMENDED that the site manager of an *ebXML Message Service* apply due diligence
1974 to the support and maintenance of its; security mechanism, site (or physical) security procedures,
1975 cryptographic protocols, update implementations and apply fixes as appropriate. (See
1976 <http://www.cert.org/> and <http://ciac.llnl.gov/>)

1977 12.2 Collaboration Protocol Agreement

1978 The configuration of Security for MSHs may be specified in the *CPA*. Three areas of the *CPA* have
1979 security definitions as follows:

- 1980 • The Document Exchange section addresses security to be applied to the payload of the message.
1981 The MSH is not responsible for any security specified at this level but may offer these services to the
1982 message sender.
- 1983 • The Message section addresses security applied to the entire ebXML Document, which includes the
1984 header and the payload.

1985 12.3 Countermeasure Technologies

1986 12.3.1 Persistent Digital Signature

1987 If signatures are being used to digitally sign an ebXML Message then XML Signature [DSIG] MUST be
1988 used to bind the ebXML SOAP **Header** and **Body** to the ebXML Payload or data elsewhere on the web
1989 that relates to the message. It is also strongly RECOMMENDED that XML Signature be used to digitally
1990 sign the Payload on its own.

1991 The only available technology that can be applied to the purpose of digitally signing an ebXML Message
1992 (the ebXML SOAP **Header** and **Body** and its associated payload objects) is provided by technology that
1993 conforms to the W3C/IETF joint XML Signature specification [XMLDSIG]. An XML Signature conforming
1994 to this specification can selectively sign portions of an XML document(s), permitting the documents to be
1995 augmented (new element content added) while preserving the validity of the signature(s).

1996 An ebXML Message requiring a digital signature SHALL be signed following the process defined in this
 1997 section of the specification and SHALL be in full compliance with [XMLDSIG].

1998 **12.3.1.1 Signature Generation**

1999 1) Create a **ds:SignedInfo** element with **ds:SignatureMethod**, **ds:CanonicalizationMethod**, and
 2000 **ds:Reference** elements for the SOAP **Header** and any required payload objects, as prescribed by
 2001 [XMLDSIG].

2002 2) Canonicalize and then calculate the **ds:SignatureValue** over **ds:SignedInfo** based on algorithms
 2003 specified in **ds:SignedInfo** as specified in [XMLDSIG].

2004 3) Construct the **ds:Signature** element that includes the **ds:SignedInfo**, **ds:KeyInfo**
 2005 (RECOMMENDED), and **ds:SignatureValue** elements as specified in [XMLDSIG].

2006 4) Include the namespace qualified **ds:Signature** element in the SOAP **Header** just signed, following
 2007 the **TraceHeaderList** element.

2008 The **ds:SignedInfo** element SHALL be composed of zero or one **ds:CanonicalizationMethod** element,
 2009 the **ds:SignatureMethod** and one or more **ds:Reference** elements.

2010 The **ds:CanonicalizationMethod** element is defined as OPTIONAL in [XMLDSIG], meaning that the
 2011 element need not appear in an instance of a **ds:SignedInfo** element. The default canonicalization
 2012 method that is applied to the data to be signed is [XMLC14N] in the absence of a **ds:Canonicalization**
 2013 element that specifies otherwise. This default SHALL also serve as the default canonicalization method
 2014 for the *ebXML Message Service*.

2015 The **ds:SignatureMethod** element SHALL be present and SHALL have an Algorithm attribute. The
 2016 RECOMMENDED value for the Algorithm attribute is:

2017 <http://www.w3.org/2000/09/xmlsig#dsa-sha1>

2018 This RECOMMENDED value SHALL be supported by all compliant *ebXML Message Service* software
 2019 implementations.

2020 The **ds:Reference** element for the SOAP **Header** document SHALL have a URI attribute value of "" to
 2021 provide for the signature to be applied to the document that contains the **ds:Signature** element (the
 2022 SOAP **Header**).

2023 The **ds:Reference** element for the SOAP **Header** MAY include a **Type** attribute that has a value
 2024 "http://www.w3.org/2000/09/xmlsig#Object" in accordance with [XMLDSIG]. This attribute is purely
 2025 informative. It MAY be omitted. Implementations of the ebXML MSH SHALL be prepared to handle
 2026 either case. The **ds:Reference** element MAY include the optional **id** attribute.

2027 The **ds:Reference** element for the SOAP **Header** SHALL include a child **ds:Transform** element. The
 2028 **ds:Transforms** element SHALL include two **ds:Transform** child elements. The first **ds:Transform**
 2029 element SHALL have a **ds:Algorithm** attribute that has a value of:

2030 <http://www.w3.org/2000/09/xmlsig#enveloped-signature>

2031 The second **ds:Transform** element SHALL have a child **ds:XPath** element that has a value of:

2032 not(ancestor-or-self::eb:TraceHeaderList or
 2033 ancestor-or-self::eb:Via)

2034 The result of the first [XPath] statement excludes the **ds:Signature** element within which it is contained,
 2035 and all its descendants, and the second [XPath] statement excludes the **TraceHeaderList** and **Via**
 2036 elements and all their descendants, as these elements are subject to change.

2037 Each payload object that requires signing SHALL be represented by a **ds:Reference** element that SHALL
 2038 have a **URI** attribute that resolves to that payload object. This MAY be either the `Content-Id` URI of the
 2039 MIME body part of the payload object, or a URI that matches the `Content-Location` of the MIME body part
 2040 of the payload object, or a URI that resolves to an external payload object external to the Message

2041 Package. It is strongly RECOMMENDED that the URI attribute value match the xlink:href URI value of the
 2042 corresponding **Manifest/Reference** element for that payload object. However, this is NOT REQUIRED.

2043 Example of digitally signed ebXML SOAP Message:

2044
 2045
 2046
 2047
 2048
 2049
 2050
 2051
 2052
 2053
 2054
 2055
 2056
 2057
 2058
 2059
 2060
 2061
 2062
 2063
 2064
 2065
 2066
 2067
 2068
 2069
 2070
 2071
 2072
 2073
 2074
 2075
 2076
 2077
 2078
 2079
 2080
 2081
 2082
 2083
 2084
 2085
 2086
 2087
 2088
 2089
 2090
 2091
 2092

```
<?xml version="1.0" encoding="utf-8"?>
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:eb="http://www.ebxml.org/namespaces/messageHeader"
  xmlns:xlink="http://www.w3.org/1999/xlink">
  <SOAP-ENV:Header>
    <eb:MessageHeader eb:id="..." eb:version="1.0">
      ...
    </eb:MessageHeader>
    <eb:TraceHeaderList eb:id="..." eb:version="1.0">
      <eb:TraceHeader>
        ...
      </eb:TraceHeader>
    </eb:TraceHeaderList>
    <ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
      <ds:SignedInfo>
        <ds:CanonicalizationMethod Algorithm="http://www.w3.org/TR/2000/CR-xml-c14n-20001026"/>
        <ds:SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#dsa-sha1"/>
        <ds:Reference URI="">
          <Transforms>
            <Transform Algorithm="http://www.w3.org/TR/1999/REC-xpath-19991116">
              <XPath xmlns:dsig="http://www.w3.org/2000/09/xmldsig#">
                not(ancestor-or-self::eb:TraceHeaderList or
                  ancestor-or-self::eb:Via)
              </XPath>
            </Transform>
          </Transforms>
          <ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#dsa-sha1"/>
          <ds:DigestValue>...</ds:DigestValue>
        </ds:Reference>
        <ds:Reference URI="cid://blahblahblah/">
          <ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#dsa-sha1"/>
          <ds:DigestValue>...</ds:DigestValue>
        </ds:Reference>
      </ds:SignedInfo>
      <ds:SignatureValue>...</ds:SignatureValue>
      <ds:KeyInfo>...</ds:KeyInfo>
    </ds:Signature>
  </SOAP-ENV:Header>
  <SOAP-ENV:Body>
    <eb:Manifest eb:id="Mani01" eb:version="1.0">
      <eb:Reference xlink:href="cid://blahblahblah"
        xlink:role="http://ebxml.org/gci/invoice">
        <eb:Schema eb:version="1.0" eb:location="http://ebxml.org/gci/busdocs/invoice.dtd"/>
      </eb:Reference>
    </eb:Manifest>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

2093 **12.3.2 Persistent Signed Receipt**

2094 An *ebXML Message* that has been digitally signed MAY be acknowledged with a **DeliveryReceipt**
 2095 acknowledgment message that itself is digitally signed in the manner described in the previous section.
 2096 The acknowledgment message MUST contain a **ds:Reference** element contained in the **ds:Signature**
 2097 element of the original message within the **Acknowledgment** element.

2098 **12.3.3 Non-persistent Authentication**

2099 Non-persistent authentication is provided by the communications channel used to transport the *ebXML*
 2100 *Message*. This authentication MAY be either in one direction, from the session initiator to the receiver, or
 2101 bi-directional. The specific method will be determined by the communications protocol used. For
 2102 instance, the use of a secure network protocol, such as [RFC2246] or [IPSEC] provides the sender of an
 2103 *ebXML Message* with a way to authenticate the destination for the TCP/IP environment.

2104 **12.3.4 Non-persistent Integrity**

2105 Use of a secure network protocol such as [RFC2246] or [IPSEC] MAY be configured to provide for
2106 integrity check CRCs of the packets transmitted via the network connection.

2107 **12.3.5 Persistent Confidentiality**

2108 XML Encryption is a W3C/IETF joint activity that is actively engaged in the drafting of a specification for
2109 the selective encryption of an XML document(s). It is anticipated that this specification will be completed
2110 within the next year. The ebXML Transport, Routing and Packaging team has identified this technology
2111 as the only viable means of providing persistent, selective confidentiality of elements within an *ebXML*
2112 *Message* including the SOAP *Header*.

2113 Confidentiality for ebXML Payloads MAY be provided by functionality possessed by a MSH. However,
2114 this specification states that it is not the responsibility of the MSH to provide security for the ebXML
2115 Payloads. Payload confidentiality MAY be provided by using XML Encryption (when available) or
2116 some other cryptographic process (such as [S/MIME], [S/MIMEV3], or [PGP/MIME]) bilaterally agreed
2117 upon by the parties involved. Since XML Encryption is not currently available, it is RECOMMENDED that
2118 [S/MIME] encryption methods be used for ebXML Payloads. The XML Encryption standard SHALL be
2119 the default encryption method when XML Encryption has achieved W3C Recommendation status.

2120 **12.3.6 Non-persistent Confidentiality**

2121 Use of a secure network protocol such as [RFC2246] or [IPSEC] provides transient confidentiality of a
2122 message as it is transferred between two ebXML MSH nodes.

2123 **12.3.7 Persistent Authorization**

2124 The OASIS Security Services Technical Committee (TC) is actively engaged in the definition of a
2125 specification that provides for the exchange of security credentials, including NameAssertion and
2126 Entitlements that is based on [SAML]. Use of technology that is based on this anticipated specification
2127 MAY be used to provide persistent authorization for an *ebXML Message* once it becomes available.
2128 ebXML has a formal liaison to this TC. There are also many ebXML member organizations and
2129 contributors that are active members of the OASIS Security Services TC such as Sun, IBM,
2130 CommerceOne, Cisco and others that are endeavoring to ensure that the specification meets the
2131 requirements of providing persistent authorization capabilities for the *ebXML Message Service*.

2132 **12.3.8 Non-persistent Authorization**

2133 Use of a secure network protocol such as [RFC2246] or [IPSEC] MAY be configured to provide for
2134 bilateral authentication of certificates prior to establishing a session. This provides for the ability for an
2135 ebXML MSH to authenticate the source of a connection that can be used to recognize the source as an
2136 authorized source of *ebXML Messages*.

2137 **12.3.9 Trusted Timestamp**

2138 At the time of this specification, services that offer trusted timestamp capabilities are becoming available.
2139 Once these become more widely available, and a standard has been defined for their use and
2140 expression, these standards, technologies and services will be evaluated and considered for use to
2141 provide this capability.

2142 **12.3.10 Supported Security Services**

2143 The general architecture of the ebXML Message Service Specification is intended to support all the
2144 security services required for electronic business. The following table combines the security services of
2145 the Message Service Handler into a set of security profiles. These profiles, or combinations of these
2146 profiles, support the specific security policy of the ebXML user community. Due to the immature state of
2147 XML security specifications, this version of the specification requires support for profiles 0 and 1 only.
2148 This does not preclude users from employing additional security features to protect ebXML exchanges;
2149 however, interoperability between parties using any profiles other than 0 and 1 cannot be guaranteed.

2150

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	✓									<i>Sending MSH</i> applies XML/DSIG structures to message
	Profile 2		✓						✓		<i>Sending MSH</i> authenticates and <i>Receiving MSH</i> authorizes sender based on communication channel credentials.
	Profile 3		✓				✓				<i>Sending MSH</i> authenticates and both MSHs negotiate a secure channel to transmit data
	Profile 4		✓		✓						<i>Sending MSH</i> authenticates, the <i>Receiving MSH</i> performs integrity checks using communications protocol
	Profile 5		✓								<i>Sending MSH</i> authenticates the communication channel only (e.g., SSL 3.0 over TCP/IP)
	Profile 6	✓					✓				<i>Sending MSH</i> applies XML/DSIG structures to message and passes in secure communications channel
	Profile 7	✓		✓							<i>Sending MSH</i> applies XML/DSIG structures to message and <i>Receiving MSH</i> 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 <i>Receiving MSH</i> returning a signed receipt
	Profile 11	✓					✓			✓	Profile 6 with the <i>Receiving MSH</i> applying a trusted timestamp
	Profile 12	✓		✓			✓			✓	Profile 8 with the <i>Receiving MSH</i> applying a trusted timestamp
	Profile 13	✓				✓					<i>Sending MSH</i> applies XML/DSIG structures to message and applies confidentiality structures (XML-Encryption)
	Profile 14	✓		✓		✓					Profile 13 with a signed receipt
	Profile 15	✓		✓						✓	<i>Sending MSH</i> applies XML/DSIG structures to message, a trusted timestamp is added to message, <i>Receiving MSH</i> returns a signed

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
											receipt
	Profile 16	✓				✓				✓	Profile 13 with a trusted timestamp applied
	Profile 17	✓		✓		✓				✓	Profile 14 with a trusted timestamp applied
	Profile 18	✓						✓			<i>Sending MSH</i> applies XML/DSIG structures to message and forwards authorization credentials [SAML]
	Profile 19	✓		✓				✓			Profile 18 with <i>Receiving MSH</i> returning a signed receipt
	Profile 20	✓		✓				✓		✓	Profile 19 with the a trusted timestamp being applied to the <i>Sending MSH</i> message
	Profile 21	✓		✓		✓		✓		✓	Profile 19 with the <i>Sending MSH</i> applying confidentiality structures (XML-Encryption)
	Profile 22					✓					<i>Sending MSH</i> encapsulates the message within confidentiality structures (XML-Encryption)

2151

2152 13 References

2153 13.1 Normative References

- 2154 [RFC2119] Key Words for use in RFCs to Indicate Requirement Levels, Internet Engineering
2155 Task Force RFC 2119, March 1997
- 2156 [HTTP] IETF RFC 2068 - Hypertext Transfer Protocol -- HTTP/1.1, R. Fielding, J. Gettys, J.
2157 Mogul, H. Frystyk, T. Berners-Lee, January 1997
- 2158 [RFC822] Standard for the Format of ARPA Internet text messages. D. Crocker. August 1982.
- 2159 [RFC2045] IETF RFC 2045. Multipurpose Internet Mail Extensions (MIME) Part One: Format of
2160 Internet Message Bodies, N Freed & N Borenstein, Published November 1996
- 2161 [RFC2046] Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types. N. Freed, N.
2162 Borenstein. November 1996.
- 2163 [RFC2246] RFC 2246 - Dierks, T. and C. Allen, "The TLS Protocol", January 1999.
- 2164 [RFC2387] The MIME Multipart/Related Content-type. E. Levinson. August 1998.
- 2165 [RFC2392] IETF RFC 2392. Content-ID and Message-ID Uniform Resource Locators. E.
2166 Levinson, Published August 1998
- 2167 [RFC2396] IETF RFC 2396. Uniform Resource Identifiers (URI): Generic Syntax. T Berners-Lee,
2168 Published August 1998
- 2169 [RFC2487] SMTP Service Extension for Secure SMTP over TLS. P. Hoffman. January 1999.
- 2170 [RFC2554] SMTP Service Extension for Authentication. J. Myers. March 1999.
- 2171 [RFC2616] RFC 2616 - Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P. and
2172 T. Berners-Lee, "Hypertext Transfer Protocol, HTTP/1.1", , June 1999.
- 2173 [RFC2617] RFC2617 - Franks, J., Hallam-Baker, P., Hostetler, J., Lawrence, S., Leach, P.,
2174 Luotonen, A., Sink, E. and L. Stewart, "HTTP Authentication: Basic and Digest
2175 Access Authentication", June 1999.
- 2176 [RFC2817] RFC 2817 - Khare, R. and S. Lawrence, "Upgrading to TLS Within HTTP/1.1", May
2177 2000.
- 2178 [RFC2818] RFC 2818 - Rescorla, E., "HTTP Over TLS", May 2000 [SOAP] Simple Object
2179 Access Protocol
- 2180 [SMTP] IETF RFC 822, Simple Mail Transfer Protocol, D Crocker, August 1982
- 2181 [SOAP] W3C-Draft-Simple Object Access Protocol (SOAP) v1.1, Don Box, DevelopMentor;
2182 David Ehnebuske, IBM; Gopal Kakivaya, Andrew Layman, Henrik Frystyk Nielsen,
2183 Satish Thatte, Microsoft; Noah Mendelsohn, Lotus Development Corp.; Dave Winer,
2184 UserLand Software, Inc.; W3C Note 08 May 2000, <http://www.w3.org/TR/SOAP>
- 2185 [SOAPATTACH] SOAP Messages with Attachments, John J. Barton, Hewlett Packard Labs; Satish
2186 Thatte and Henrik Frystyk Nielsen, Microsoft, Published Oct 09 2000
2187 <http://www.w3.org/TR/SOAP-attachments>
- 2188 [SSL3] A. Frier, P. Karlton, and P. Kocher, "The SSL 3.0 Protocol", Netscape
2189 Communications Corp., Nov 18, 1996.
- 2190 [UTF-8] UTF-8 is an encoding that conforms to ISO/IEC 10646. See [XML] for usage
2191 conventions.
- 2192 [XLINK] W3C XML Linking Candidate Recommendation, <http://www.w3.org/TR/xlink/>

- 2193 [XML] W3C Recommendation: Extensible Markup Language (XML) 1.0 (Second Edition),
2194 October 2000, <http://www.w3.org/TR/2000/REC-xml-20001006>
- 2195 [XML Namespace] W3C Recommendation for Namespaces in XML, World Wide Web Consortium, 14
2196 January 1999, <http://www.w3.org/TR/REC-xml-names>
- 2197 [XMLDSIG] Joint W3C/IETF XML-Signature Syntax and Processing specification,
2198 <http://www.w3.org/TR/2000/CR-xmlsig-core-20001031/>
- 2199 [XMLMedia] IETF RFC 3023, XML Media Types. M. Murata, S. St.Laurent, January 2001
- 2200 **13.2 Non-Normative References**
- 2201 [ebCPP] ebXML Collaboration Protocol Profile and Agreement specification, Version 1.0,
2202 published 11 May, 2001
- 2203 [ebBPSS] ebXML Business Process Specification Schema, version 1.0, published 27 April
2204 2001.
- 2205 [ebTA] ebXML Technical Architecture, version 1.04 published 16 February, 2001
- 2206 [ebTASEC] ebXML Technical Architecture Risk Assessment Technical Report, version 0.36
2207 published 20 April 2001
- 2208 [ebRS] ebXML Registry Services Specification, version 0.84
- 2209 [ebMSREQ] ebXML Transport, Routing and Packaging: Overview and Requirements, Version
2210 0.96, Published 25 May 2000
- 2211 [ebGLOSS] ebXML Glossary, <http://www.ebxml.org>, published 11 May, 2001.
- 2212 [IPSEC] IETF RFC2402 IP Authentication Header. S. Kent, R. Atkinson. November 1998.
2213 RFC2406 IP Encapsulating Security Payload (ESP). S. Kent, R. Atkinson. November
2214 1998.
- 2215 [PGP/MIME] IETF RFC2015, "MIME Security with Pretty Good Privacy (PGP)", M. Elkins. October
2216 1996.
- 2217 [SAML] Security Assertion Markup Language,
2218 <http://www.oasis-open.org/committees/security/docs/draft-sstc-use-strawman-03.html>
- 2219 [S/MIME] IETF RFC2311, "S/MIME Version 2 Message Specification", S. Dusse, P. Hoffman,
2220 B. Ramsdell, L. Lundblade, L. Repka. March 1998.
- 2221 [S/MIMECH] IETF RFC 2312, "S/MIME Version 2 Certificate Handling", S. Dusse, P. Hoffman, B.
2222 Ramsdell, J. Weinstein. March 1998.
- 2223 [S/MIMEV3] IETF RFC 2633 S/MIME Version 3 Message Specification. B. Ramsdell, Ed.. June
2224 1999.
- 2225 [TLS] RFC2246, T. Dierks, C. Allen. January 1999.
- 2226 [XMLSchema] W3C XML Schema Candidate Recommendation,
2227 <http://www.w3.org/TR/xmlschema-0/>
2228 <http://www.w3.org/TR/xmlschema-1/>
2229 <http://www.w3.org/TR/xmlschema-2/>
- 2230 [XMTP] XMTP - Extensible Mail Transport Protocol
2231 <http://www.openhealth.org/documents/xmtp.htm>

2232 **14 Contact Information**

2233 **Team Leader**

2234 Name Rik Drummond
 2235 Company Drummond Group, Inc.
 2236 Street 5008 Bentwood Ct.
 2237 City, State, Postal Code Fort Worth, Texas 76132
 2238 Country USA
 2239 Phone +1 (817) 294-7339
 2240 EMail: rik@drummondgroup.com

2241

2242 **Vice Team Leader**

2243 Name Christopher Ferris
 2244 Company Sun Microsystems
 2245 Street One Network Drive
 2246 City, State, Postal Code Burlington, MA 01803-0903
 2247 Country USA
 2248 Phone: +1 (781) 442-3063
 2249 EMail: chris.ferris@sun.com

2250

2251 **Team Editor**

2252 Name David Burdett
 2253 Company Commerce One
 2254 Street 4400 Rosewood Drive
 2255 City, State, Postal Code Pleasanton, CA 94588
 2256 Country USA
 2257 Phone: +1 (925) 520-4422
 2258 EMail: david.burdett@commerceone.com

2259

2260 **Authors**

2261 Name Dick Brooks
 2262 Company Group 8760
 2263 Street 110 12th Street North, Suite F103
 2264 City, State, Postal Code Birmingham, Alabama 35203
 2265 Phone: +1 (205) 250-8053
 2266 Email: dick@8760.com

2267

2268 Name David Burdett
 2269 Company Commerce One
 2270 Street 4400 Rosewood Drive
 2271 City, State, Postal Code Pleasanton, CA 94588
 2272 Country USA
 2273 Phone: +1 (925) 520-4422
 2274 EMail: david.burdett@commerceone.com

2275

2276 Name Christopher Ferris
 2277 Company Sun Microsystems
 2278 Street One Network Drive
 2279 City, State, Postal Code Burlington, MA 01803-0903
 2280 Country USA
 2281 Phone: +1 (781) 442-3063
 2282 EMail: chris.ferris@east.sun.com

2283

2284 Name John Ibbotson
 2285 Company IBM UK Ltd

2286 Street Hursley Park
 2287 City, State, Postal Code Winchester SO21 2JN
 2288 Country United Kingdom
 2289 Phone: +44 (1962) 815188
 2290 Email: john_ibbotson@uk.ibm.com
 2291
 2292 Name Masayoshi Shimamura
 2293 Company Fujitsu Limited
 2294 Street Shinyokohama Nikko Bldg., 15-16, Shinyokohama 2-chome
 2295 City, State, Postal Code Kohoku-ku, Yokohama 222-0033, Japan
 2296 Phone: +81-45-476-4590
 2297 EMail: shima@rp.open.cs.fujitsu.co.jp
 2298
2299 Document Editing Team
 2300 Name Ralph Berwanger
 2301 Company bTrade.com
 2302 Street 2324 Gateway Drive
 2303 City, State, Postal Code Irving, TX 75063
 2304 Country USA
 2305 Phone: +1 (972) 580-3970
 2306 EMail: rberwanger@btrade.com
 2307
 2308 Name Colleen Evans
 2309 Company Progress/Sonic Software
 2310 Street 14 Oak Park
 2311 City,State,Postal Code Bedford, MA 01730
 2312 Country USA
 2313 Phone +1 (720) 480-3919
 2314 Email cevans@progress.com
 2315
 2316 Name Ian Jones
 2317 Company British Telecommunications
 2318 Street Enterprise House, 84-85 Adam Street
 2319 City, State, Postal Code Cardiff, CF24 2XF
 2320 Country United Kingdom
 2321 Phone: +44 29 2072 4063
 2322 EMail: ian.c.jones@bt.com
 2323
 2324 Name Martha Warfelt
 2325 Company DaimlerChrysler Corporation
 2326 Street 800 Chrysler Drive
 2327 City, State, Postal Code Auburn Hills, MI
 2328 Country USA
 2329 Phone: +1 (248) 944-5481
 2330 EMail: maw2@daimlerchrysler.com
 2331
 2332 Name David Fischer
 2333 Company Drummond Group, Inc
 2334 Street 5008 Bentwood Ct
 2335 City, State, Postal Code Fort Worth, TX 76132
 2336 Phone +1 (817-294-7339
 2337 EMail david@drummondgroup.com

2338 **Appendix A ebXML SOAP Extension Elements Schema**

2339 The ebXML SOAP extension elements schema has been specified using the Candidate
 2340 Recommendation draft of the XML Schema specification[XMLSchema]. Because ebXML has adopted
 2341 SOAP 1.1 for the message format, and because the SOAP 1.1 schema resolved by the SOAP 1.1
 2342 namespace URI was written to an earlier draft of the XML Schema specification, the ebXML TRP team
 2343 has created a version of the SOAP 1.1 envelope schema that is specified using the schema vocabulary
 2344 that conforms to the W3C XML Schema Candidate Recommendation specification[XMLSchema].

2345 In addition, it was necessary to craft a schema for the [XLINK] attribute vocabulary and for the XML
 2346 xml:lang attribute.

2347 Finally, because certain authoring tools do not correctly resolve local entities when importing schema, a
 2348 version of the W3C XML Signature Core schema has also been provided and referenced by the ebXML
 2349 SOAP extension elements schema defined in this Appendix.

2350 These alternative schema SHALL be available from the following URL's:

2351 XML Signature Core – http://ebxml.org/project_teams/transport/xmldsig-core-schema.xsd

2352 Xlink - http://ebxml.org/project_teams/transport/xlink.xsd

2353 xml:lang - http://ebxml.org/project_teams/transport/xml_lang.xsd

2354 SOAP1.1 - http://ebxml.org/project_teams/transport/envelope.xsd

2355 Note: if inconsistencies exist between the specification and this schema, the specification supersedes this example schema.

```

2356 <?xml version="1.0" encoding="UTF-8"?>
2357 <schema targetNamespace="http://www.ebxml.org/namespaces/messageHeader"
2358 xmlns:xml="http://www.w3.org/XML/1998/namespace"
2359 xmlns:tns="http://www.ebxml.org/namespaces/messageHeader" xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
2360 xmlns:xlink="http://www.w3.org/1999/xlink" xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
2361 xmlns="http://www.w3.org/2000/10/XMLSchema" version="1.0">
2362 <import namespace="http://www.w3.org/2000/09/xmldsig#"
2363 schemaLocation="http://www.ebxml.org/project_teams/transport/xmldsig-core-schema.xsd"/>
2364 <import namespace="http://www.w3.org/1999/xlink"
2365 schemaLocation="http://www.ebxml.org/project_teams/transport/xlink.xsd"/>
2366 <import namespace="http://schemas.xmlsoap.org/soap/envelope/"
2367 schemaLocation="http://www.ebxml.org/project_teams/transport/envelope.xsd"/>
2368 <import namespace="http://www.w3.org/XML/1998/namespace"
2369 schemaLocation="http://www.ebxml.org/project_teams/transport/xml_lang.xsd"/>
2370 <!-- MANIFEST -->
2371 <element name="Manifest">
2372 <complexType>
2373 <sequence>
2374 <element ref="tns:Reference" maxOccurs="unbounded"/>
2375 <any namespace="##other" processContents="lax" minOccurs="0" maxOccurs="unbounded"/>
2376 </sequence>
2377 <attribute ref="tns:id"/>
2378 <attribute ref="tns:version"/>
2379 <anyAttribute namespace="http://www.w3.org/2000/10/XMLSchema-instance"
2380 processContents="lax"/>
2381 </complexType>
2382 </element>
2383 <element name="Reference">
2384 <complexType>
2385 <sequence>
2386 <element ref="tns:Schema" minOccurs="0" maxOccurs="unbounded"/>
2387 <element ref="tns:Description" minOccurs="0" maxOccurs="unbounded"/>
2388 <any namespace="##other" processContents="lax" minOccurs="0" maxOccurs="unbounded"/>
2389 </sequence>
2390 <attribute ref="tns:id"/>
2391 <attribute ref="xlink:type" use="fixed" value="simple"/>
2392 <attribute ref="xlink:href" use="required"/>
2393
    
```

```

2394     <attribute ref="xlink:role" />
2395   </complexType>
2396 </element>
2397 <element name="Schema">
2398   <complexType>
2399     <attribute name="location" type="uriReference" use="required" />
2400     <attribute name="version" type="tns:non-empty-string" />
2401   </complexType>
2402 </element>
2403 <!-- MESSAGEHEADER -->
2404 <element name="MessageHeader">
2405   <complexType>
2406     <sequence>
2407       <element ref="tns:From" />
2408       <element ref="tns:To" />
2409       <element ref="tns:CPAId" />
2410       <element ref="tns:ConversationId" />
2411       <element ref="tns:Service" />
2412       <element ref="tns:Action" />
2413       <element ref="tns:MessageData" />
2414       <element ref="tns:QualityOfServiceInfo" minOccurs="0" />
2415       <element ref="tns:Description" minOccurs="0" maxOccurs="unbounded" />
2416       <element ref="tns:SequenceNumber" minOccurs="0" />
2417     </sequence>
2418     <attribute ref="tns:id" />
2419     <attribute ref="tns:version" />
2420     <attribute ref="soap:mustUnderstand" />
2421     <anyAttribute namespace="http://www.w3.org/2000/10/XMLSchema-instance"
2422       processContents="lax" />
2423   </complexType>
2424 </element>
2425 <element name="CPAId" type="tns:non-empty-string" />
2426 <element name="ConversationId" type="tns:non-empty-string" />
2427 <element name="Service">
2428   <complexType>
2429     <simpleContent>
2430       <extension base="tns:non-empty-string">
2431         <attribute name="type" type="tns:non-empty-string" />
2432       </extension>
2433     </simpleContent>
2434   </complexType>
2435 </element>
2436 <element name="Action" type="tns:non-empty-string" />
2437 <element name="MessageData">
2438   <complexType>
2439     <sequence>
2440       <element ref="tns:MessageId" />
2441       <element ref="tns:Timestamp" />
2442       <element ref="tns:RefToMessageId" minOccurs="0" />
2443       <element ref="tns:TimeToLive" minOccurs="0" />
2444     </sequence>
2445   </complexType>
2446 </element>
2447 <element name="MessageId" type="tns:non-empty-string" />
2448 <element name="TimeToLive" type="timeInstant" />
2449 <element name="QualityOfServiceInfo">
2450   <complexType>
2451     <attribute name="deliverySemantics" type="tns:deliverySemantics.type" use="default"
2452       value="BestEffort" />
2453     <attribute name="messageOrderSemantics" type="tns:messageOrderSemantics.type"
2454       use="default" value="NotGuaranteed" />
2455     <attribute name="deliveryReceiptRequested" type="tns:signedUnsigned.type"
2456       use="default" value="None" />
2457   </complexType>
2458 </element>
2459 <!-- TRACE HEADER LIST -->
2460 <element name="TraceHeaderList">
2461   <complexType>
2462     <sequence>
2463       <element ref="tns:TraceHeader" maxOccurs="unbounded" />
2464     </sequence>

```

```

2465     <attribute ref="tns:id"/>
2466     <attribute ref="tns:version"/>
2467     <attribute ref="soap:mustUnderstand" use="required"/>
2468     <attribute ref="soap:actor" use="required"/>
2469     <anyAttribute namespace="http://www.w3.org/2000/10/XMLSchema-instance"
2470       processContents="lax"/>
2471   </complexType>
2472 </element>
2473 <element name="TraceHeader">
2474   <complexType>
2475     <sequence>
2476       <element ref="tns:Sender"/>
2477       <element ref="tns:Receiver"/>
2478       <element ref="tns:Timestamp"/>
2479       <any namespace="##other" processContents="lax" minOccurs="0" maxOccurs="unbounded"/>
2480     </sequence>
2481     <attribute ref="tns:id"/>
2482   </complexType>
2483 </element>
2484 <element name="Sender" type="tns:senderReceiver.type"/>
2485 <element name="Receiver" type="tns:senderReceiver.type"/>
2486 <element name="SequenceNumber" type="positiveInteger"/>
2487 <!-- DELIVERY RECEIPT -->
2488 <element name="DeliveryReceipt">
2489   <complexType>
2490     <sequence>
2491       <element ref="tns:Timestamp"/>
2492       <element ref="ds:Reference" minOccurs="0" maxOccurs="unbounded"/>
2493     </sequence>
2494     <attribute ref="tns:id"/>
2495     <attribute ref="tns:version"/>
2496     <anyAttribute namespace="http://www.w3.org/2000/10/XMLSchema-instance"
2497       processContents="lax"/>
2498     <!-- <attribute name="signed" type="boolean"/> -->
2499   </complexType>
2500 </element>
2501 <!-- ACKNOWLEDGEMENT -->
2502 <element name="Acknowledgment">
2503   <complexType>
2504     <sequence>
2505       <element ref="tns:Timestamp"/>
2506       <element ref="tns:From" minOccurs="0"/>
2507       <element ref="ds:Reference" minOccurs="0" maxOccurs="unbounded"/>
2508     </sequence>
2509     <attribute ref="tns:id"/>
2510     <attribute ref="tns:version"/>
2511     <attribute ref="soap:mustUnderstand" use="required"/>
2512     <attribute ref="soap:actor" use="required"/>
2513     <anyAttribute namespace="http://www.w3.org/2000/10/XMLSchema-instance"
2514       processContents="lax"/>
2515   </complexType>
2516 </element>
2517 <!-- ERROR LIST -->
2518 <element name="ErrorList">
2519   <complexType>
2520     <sequence>
2521       <element ref="tns:Error" maxOccurs="unbounded"/>
2522     </sequence>
2523     <attribute ref="tns:id"/>
2524     <attribute ref="tns:version"/>
2525     <attribute ref="soap:mustUnderstand" use="required"/>
2526     <attribute name="highestSeverity" type="tns:severity.type"
2527       use="default" value="Warning"/>
2528     <anyAttribute namespace="http://www.w3.org/2000/10/XMLSchema-instance"
2529       processContents="lax"/>
2530   </complexType>
2531 </element>
2532 <element name="Error">
2533   <complexType>
2534     <attribute ref="tns:id"/>
2535     <attribute name="codeContext" type="uriReference" use="required"/>

```



```

2536     <attribute name="errorCode" type="tns:non-empty-string" use="required" />
2537     <attribute name="severity" type="tns:severity.type" use="default" value="Warning" />
2538     <attribute name="location" type="tns:non-empty-string" />
2539     <attribute ref="xml:lang" />
2540   </complexType>
2541 </element>
2542 <!-- STATUS RESPONSE -->
2543 <element name="StatusResponse">
2544   <complexType>
2545     <sequence>
2546       <element ref="tns:RefToMessageId" />
2547       <element ref="tns:Timestamp" minOccurs="0" />
2548     </sequence>
2549     <attribute ref="tns:id" />
2550     <attribute ref="tns:version" />
2551     <attribute name="messageStatus" type="tns:messageStatus.type" />
2552     <anyAttribute namespace="http://www.w3.org/2000/10/XMLSchema-instance"
2553       processContents="lax" />
2554   </complexType>
2555 </element>
2556 <!-- STATUS REQUEST -->
2557 <element name="StatusRequest">
2558   <complexType>
2559     <sequence>
2560       <element ref="tns:RefToMessageId" />
2561     </sequence>
2562     <attribute ref="tns:id" />
2563     <attribute ref="tns:version" />
2564     <anyAttribute namespace="http://www.w3.org/2000/10/XMLSchema-instance"
2565       processContents="lax" />
2566   </complexType>
2567 </element>
2568 <!-- VIA -->
2569 <element name="Via">
2570   <complexType>
2571     <sequence>
2572       <element ref="tns:CPAId" minOccurs="0" />
2573       <element ref="tns:Service" minOccurs="0" />
2574       <element ref="tns:Action" minOccurs="0" />
2575     </sequence>
2576     <attribute ref="tns:id" />
2577     <attribute ref="tns:version" />
2578     <attribute ref="soap:mustUnderstand" use="required" />
2579     <attribute ref="soap:actor" use="required" />
2580     <attribute name="syncReply" type="boolean" />
2581     <attribute name="deliveryReceiptRequested" type="tns:signedUnsigned.type"
2582       use="default" value="None" />
2583     <attribute name="reliableMessagingMethod" type="tns:rmm.type" />
2584     <attribute name="ackRequested" type="boolean" />
2585     <anyAttribute namespace="http://www.w3.org/2000/10/XMLSchema-instance"
2586       processContents="lax" />
2587   </complexType>
2588 </element>
2589 <!-- COMMON TYPES -->
2590 <complexType name="senderReceiver.type">
2591   <sequence>
2592     <element ref="tns:PartyId" maxOccurs="unbounded" />
2593     <element name="Location" type="uriReference" />
2594   </sequence>
2595 </complexType>
2596 <simpleType name="messageStatus.type">
2597   <restriction base="NMTOKEN">
2598     <enumeration value="Unauthorized" />
2599     <enumeration value="NotRecognized" />
2600     <enumeration value="Received" />
2601     <enumeration value="Processed" />
2602     <enumeration value="Forwarded" />
2603   </restriction>
2604 </simpleType>
2605 <simpleType name="type.type">
2606   <restriction base="NMTOKEN">

```

```

2607     <enumeration value="DeliveryReceipt" />
2608     <enumeration value="IntermediateAck" />
2609   </restriction>
2610 </simpleType>
2611 <simpleType name="messageOrderSemantics.type">
2612   <restriction base="NMTOKEN">
2613     <enumeration value="Guaranteed" />
2614     <enumeration value="NotGuaranteed" />
2615   </restriction>
2616 </simpleType>
2617 <simpleType name="deliverySemantics.type">
2618   <restriction base="NMTOKEN">
2619     <enumeration value="OnceAndOnlyOnce" />
2620     <enumeration value="BestEffort" />
2621   </restriction>
2622 </simpleType>
2623 <simpleType name="non-empty-string">
2624   <restriction base="string">
2625     <minLength value="1" />
2626   </restriction>
2627 </simpleType>
2628 <simpleType name="rmm.type">
2629   <restriction base="NMTOKEN">
2630     <enumeration value="ebXML" />
2631     <enumeration value="Transport" />
2632   </restriction>
2633 </simpleType>
2634 <simpleType name="signedUnsigned.type">
2635   <restriction base="NMTOKEN">
2636     <enumeration value="Signed" />
2637     <enumeration value="Unsigned" />
2638     <enumeration value="None" />
2639   </restriction>
2640 </simpleType>
2641 <simpleType name="severity.type">
2642   <restriction base="NMTOKEN">
2643     <enumeration value="Warning" />
2644     <enumeration value="Error" />
2645   </restriction>
2646 </simpleType>
2647 <!-- COMMON ATTRIBUTES and ELEMENTS -->
2648 <attribute name="id" type="ID" form="unqualified" />
2649 <attribute name="version" type="tns:non-empty-string" use="fixed" value="1.0" />
2650 <element name="PartyId">
2651   <complexType>
2652     <simpleContent>
2653       <extension base="tns:non-empty-string">
2654         <attribute name="type" type="tns:non-empty-string" />
2655       </extension>
2656     </simpleContent>
2657   </complexType>
2658 </element>
2659 <element name="To">
2660   <complexType>
2661     <sequence>
2662       <element ref="tns:PartyId" maxOccurs="unbounded" />
2663     </sequence>
2664   </complexType>
2665 </element>
2666 <element name="From">
2667   <complexType>
2668     <sequence>
2669       <element ref="tns:PartyId" maxOccurs="unbounded" />
2670     </sequence>
2671   </complexType>
2672 </element>
2673 <element name="Description">
2674   <complexType>
2675     <simpleContent>
2676       <extension base="tns:non-empty-string">
2677         <attribute ref="xml:lang" />

```

```
2678     </extension>
2679     </simpleContent>
2680   </complexType>
2681 </element>
2682 <element name="RefToMessageId" type="tns:non-empty-string"/>
2683 <element name="Timestamp" type="timeInstant"/>
2684 </schema>
2685
```

2686 Appendix B Communication Protocol Bindings

2687 B.1 Introduction

2688 One of the goals of ebXML's Transport, Routing and Packaging team is to design a message handling
2689 service usable over a variety of network and application level communication protocols. These protocols
2690 serve as the "carrier" of ebXML Messages and provide the underlying services necessary to carry out a
2691 complete ebXML Message exchange between two parties. HTTP, FTP, Java Message Service (JMS)
2692 and SMTP are examples of application level communication protocols. TCP and SNA/LU6.2 are
2693 examples of network transport protocols. Communication protocols vary in their support for data content,
2694 processing behavior and error handling and reporting. For example, it is customary to send binary data in
2695 raw form over HTTP. However, in the case of SMTP it is customary to "encode" binary data into a 7-bit
2696 representation. HTTP is equally capable of carrying out *synchronous* or *asynchronous* message
2697 exchanges whereas it is likely that message exchanges occurring over SMTP will be *asynchronous*. This
2698 section describes the technical details needed to implement this abstract ebXML Message Handling
2699 Service over particular communication protocols.

2700 This section specifies communication protocol bindings and technical details for carrying *ebXML Message*
2701 *Service* messages for the following communication protocols:

- 2702 • Hypertext Transfer Protocol [HTTP], in both *asynchronous* and *synchronous* forms of transfer.
- 2703 • Simple Mail Transfer Protocol [SMTP], in *asynchronous* form of transfer only.

2704 B.2 HTTP

2705 B.2.1 Minimum level of HTTP protocol

2706 Hypertext Transfer Protocol Version 1.1 [HTTP] (<http://www.ietf.org/rfc2616.txt>) is the minimum level of
2707 protocol that MUST be used.

2708 B.2.2 Sending ebXML Service messages over HTTP

2709 Even though several HTTP request methods are available, this specification only defines the use of HTTP
2710 POST requests for sending *ebXML Message Service* messages over HTTP. The identity of the ebXML
2711 MSH (e.g. ebxmlhandler) may be part of the HTTP POST request:

2712
2713

```
POST /ebxmlhandler HTTP/1.1
```

2714 Prior to sending over HTTP, an ebXML Message MUST be formatted according to ebXML Message
2715 Service Specification sections 7 and 8. Additionally, the messages MUST conform to the HTTP specific
2716 MIME canonical form constraints specified in section 19.4 of RFC 2616 [HTTP] specification (see:
2717 <http://www.ietf.org/rfc2616.txt>).

2718 HTTP protocol natively supports 8-bit and Binary data. Hence, transfer encoding is OPTIONAL for such
2719 parts in an ebXML Service Message prior to sending over HTTP. However, content-transfer-encoding of
2720 such parts (e.g. using base64 encoding scheme) is not precluded by this specification.

2721 The rules for forming an HTTP message containing an ebXML Service Message are as follows:

- 2722 • The **Content-Type: multipart/related** MIME header with the associated parameters, from the
2723 ebXML Service Message Envelope MUST appear as an HTTP header.
- 2724 • All other MIME headers that constitute the ebXML Message Envelope MUST also become part of the
2725 HTTP header.

- 2726 • The mandatory SOAPAction HTTP header field must also be included in the HTTP header and MAY
2727 have a value of "ebXML"

2728 SOAPAction: "ebXML"

- 2729 • Other headers with semantics defined by MIME specifications, such as Content-Transfer-Encoding,
2730 SHALL NOT appear as HTTP headers. Specifically, the "MIME-Version: 1.0" header MUST NOT
2731 appear as an HTTP header. However, HTTP-specific MIME-like headers defined by HTTP 1.1 MAY
2732 be used with the semantic defined in the HTTP specification.
- 2733 • All ebXML Service Message parts that follow the ebXML Message Envelope, including the MIME
2734 boundary string, constitute the HTTP entity body. This encompasses the SOAP **Envelope** and the
2735 constituent ebXML parts and attachments including the trailing MIME boundary strings.

2736 The example below shows an example instance of an HTTP POST'ed ebXML Service Message:

2737
2738
2739
2740
2741
2742
2743
2744
2745
2746
2747
2748
2749
2750
2751
2752
2753
2754
2755
2756
2757
2758
2759
2760
2761
2762
2763
2764
2765
2766
2767
2768
2769
2770
2771
2772
2773
2774
2775
2776
2777
2778
2779
2780
2781
2782
2783
2784
2785
2786
2787
2788
2789
2790

```
POST /servlet/ebXMLhandler HTTP/1.1
Host: www.example2.com
SOAPAction: "ebXML"
Content-type: multipart/related; boundary="Boundary"; type="text/xml";
      start=" <ebxhmheader111@example.com>"
--Boundary
Content-ID: <ebxhmheader111@example.com>
Content-Type: text/xml

<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV='http://schemas.xmlsoap.org/soap/envelope/'
  xmlns:eb='http://www.ebxml.org/namespaces/messageHeader'>
<SOAP-ENV:Header>
  <eb:MessageHeader SOAP-ENV:mustUnderstand="1" eb:version="1.0">
    <eb:From>
      <eb:PartyId>urn:duns:123456789</eb:PartyId>
    </eb:From>
    <eb:To>
      <eb:PartyId>urn:duns:912345678</eb:PartyId>
    </eb:To>
    <eb:CPAId>20001209-133003-28572</eb:CPAId>
    <eb:ConversationId>20001209-133003-28572</eb:ConversationId>
    <eb:Service>urn:services:SupplierOrderProcessing</eb:Service>
    <eb:Action>NewOrder</eb:Action>
    <eb:MessageData>
      <eb:MessageId>20001209-133003-28572@example.com</eb:MessageId>
      <eb:Timestamp>2001-02-15T11:12:12Z</Timestamp>
    </eb:MessageData>
    <eb:QualityOfServiceInfo eb:deliverySemantics="BestEffort"/>
  </eb:MessageHeader>
</SOAP-ENV:Header>
<SOAP-ENV:Body>
  <eb:Manifest SOAP-ENV:mustUnderstand="1" eb:version="1.0">
    <eb:Reference xlink:href="cid:ebxmlpayload111@example.com"
      xlink:role="XLinkRole"
      xlink:type="simple">
      <eb:Description xml:lang="en-us">Purchase Order 1</eb:Description>
    </eb:Reference>
  </eb:Manifest>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>

--Boundary
Content-ID: <ebxmlpayload111@example.com>
Content-Type: text/xml

<?xml version="1.0" encoding="UTF-8"?>
<purchase_order>
  <po_number>1</po_number>
  <part_number>123</part_number>
  <price currency="USD">500.00</price>
</purchase_order>
```

2791
2792

--Boundary--

2793 B.2.3 HTTP Response Codes

2794 In general, semantics of communicating over HTTP as specified in the [RFC2616] MUST be followed, for
2795 returning the HTTP level response codes. A 2xx code MUST be returned when the HTTP Posted
2796 message is successfully received by the receiving HTTP entity. However, see exception for SOAP error
2797 conditions below. Similarly, other HTTP codes in the 3xx, 4xx, 5xx range MAY be returned for conditions
2798 corresponding to them. However, error conditions encountered while processing an ebXML Service
2799 Message MUST be reported using the error mechanism defined by the ebXML Message Service
2800 Specification (see section 11).

2801 B.2.4 SOAP Error conditions and Synchronous Exchanges

2802 The SOAP 1.1 specification states:

2803 *"In case of a SOAP error while processing the request, the SOAP HTTP server MUST issue an HTTP 500*
2804 *"Internal Server Error" response and include a SOAP message in the response containing a SOAP Fault*
2805 *element indicating the SOAP processing error. "*

2806 However, the scope of the SOAP 1.1 specification is limited to *synchronous* mode of message exchange
2807 over HTTP, whereas the ebXML Message Service Specification specifies both *synchronous* and
2808 *asynchronous* modes of message exchange over HTTP. Hence, the SOAP 1.1 specification MUST be
2809 followed for *synchronous* mode of message exchange, where the SOAP *Message* containing a SOAP
2810 **Fault** element indicating the SOAP processing error MUST be returned in the HTTP response with a
2811 response code of "HTTP 500 Internal Server Error". When *asynchronous* mode of message exchange is
2812 being used, a HTTP response code in the range 2xx MUST be returned when the message is received
2813 successfully and any error conditions (including SOAP errors) must be returned via a separate HTTP
2814 Post.

2815 B.2.5 Synchronous vs. Asynchronous

2816 When the **syncReply** parameter in the *Via* element is set to "true", the response message(s) MUST be
2817 returned on the same HTTP connection as the inbound request, with an appropriate HTTP response
2818 code, as described above. When the **syncReply** parameter is set to "false", the response messages are
2819 not returned on the same HTTP connection as the inbound request, but using an independent HTTP Post
2820 request. An HTTP response with a response code as defined in section B.2.3 above and with an empty
2821 HTTP body MUST be returned in response to the HTTP Post.

2822 B.2.6 Access Control

2823 Implementers MAY protect their ebXML Message Service Handlers from unauthorized access through the
2824 use of an access control mechanism. The HTTP access authentication process described in "HTTP
2825 Authentication: Basic and Digest Access Authentication" [RFC2617] defines the access control
2826 mechanisms allowed to protect an ebXML Message Service Handler from unauthorized access.

2827 Implementers MAY support all of the access control schemes defined in [RFC2617] however they MUST
2828 support the Basic Authentication mechanism, as described in section 2, when Access Control is used.

2829 Implementers that use basic authentication for access control SHOULD also use communication protocol
2830 level security, as specified in the section titled "Confidentiality and Communication Protocol Level
2831 Security" in this document.

2832 **B.2.7 Confidentiality and Communication Protocol Level Security**

2833 An ebXML Message Service Handler MAY use transport layer encryption to protect the confidentiality of
2834 ebXML Messages and HTTP transport headers. The IETF Transport Layer Security specification
2835 [RFC2246] provides the specific technical details and list of allowable options, which may be used by
2836 ebXML Message Service Handlers. ebXML Message Service Handlers MUST be capable of operating in
2837 backwards compatibility mode with SSL [SSL3], as defined in Appendix E of [RFC2246].

2838 ebXML Message Service Handlers MAY use any of the allowable encryption algorithms and key sizes
2839 specified within [RFC2246]. At a minimum ebXML Message Service Handlers MUST support the key
2840 sizes and algorithms necessary for backward compatibility with [SSL3].

2841 The use of 40-bit encryption keys/algorithms is permitted, however it is RECOMMENDED that stronger
2842 encryption keys/algorithms SHOULD be used.

2843 Both [RFC2246] and [SSL3] require the use of server side digital certificates. In addition client side
2844 certificate based authentication is also permitted. ebXML Message Service handlers MUST support
2845 hierarchical and peer-to-peer trust models.

2846 **B.3 SMTP**

2847 The Simple Mail Transfer Protocol [SMTP] and its companion documents [RFC822] and [ESMTP]
2848 makeup the suite of specifications commonly referred to as Internet Electronic Mail. These specifications
2849 have been augmented over the years by other specifications, which define additional functionality
2850 "layered on top" of these baseline specifications. These include:

- 2851 • Multipurpose Internet Mail Extensions (MIME) [RFC2045], [RFC2046], [RFC2387]
- 2852 • SMTP Service Extension for Authentication [RFC2554]
- 2853 • SMTP Service Extension for Secure SMTP over TLS [RFC2487]

2854 Typically, Internet Electronic Mail Implementations consist of two "agent" types:

- 2855 • Message Transfer Agent (MTA): Programs that send and receive mail messages with other
2856 MTA's on behalf of MUA's. Microsoft Exchange Server is an example of a MTA
- 2857 • Mail User Agent (MUA): Electronic Mail programs are used to construct electronic mail messages
2858 and communicate with an MTA to send/retrieve mail messages. Microsoft Outlook is an example
2859 of a MUA.

2860 MTA's often serve as "mail hubs" and can typically service hundreds or more MUA's.

2861 MUA's are responsible for constructing electronic mail messages in accordance with the Internet
2862 Electronic Mail Specifications identified above. This section describes the "binding" of an ebXML
2863 compliant message for transport via eMail from the perspective of a MUA. No attempt is made to define
2864 the binding of an ebXML Message exchange over SMTP from the standpoint of a MTA.

2865 **B.3.1 Minimum level of supported protocols**

- 2866 • Simple Mail Transfer Protocol [RFC821] and [RFC822]
- 2867 • MIME [RFC2045] and [RFC2046]
- 2868 • Multipart/Related MIME [RFC2387]

2869 **B.3.2 Sending ebXML Messages over SMTP**

2870 Prior to sending messages over SMTP an ebXML Message MUST be formatted according to ebXML
 2871 Message Service Specification sections 7 and 8. Additionally the messages must also conform to the
 2872 syntax, format and encoding rules specified by MIME [RFC2045], [RFC2046] and [RFC2387].

2873 Many types of data that a party might desire to transport via email are represented as 8bit characters or
 2874 binary data. Such data cannot be transmitted over SMTP[SMTP], which restricts mail messages to 7bit
 2875 US-ASCII data with lines no longer than 1000 characters including any trailing CRLF line separator. If a
 2876 sending Message Service Handler knows that a receiving MTA, or ANY intermediary MTA's, are
 2877 restricted to handling 7-bit data then any document part that uses 8 bit (or binary) representation must be
 2878 "transformed" according to the encoding rules specified in section 6 of [RFC2045]. In cases where a
 2879 Message Service Handler knows that a receiving MTA and ALL intermediary MTA's are capable of
 2880 handling 8-bit data then no transformation is needed on any part of the ebXML Message.

2881 The rules for forming an ebXML Message for transport via SMTP are as follows:

- 2882 • If using [RFC821] restricted transport paths, apply transfer encoding to all 8-bit data that will be
 2883 transported in an ebXML message, according to the encoding rules defined in section 6 of
 2884 [RFC2045]. The Content-Transfer-Encoding MIME header MUST be included in the MIME envelope
 2885 portion of any body part that has been transformed (encoded).
- 2886 • The Content-Type: Multipart/Related MIME header with the associated parameters, from the
 2887 ebXML Message Envelope MUST appear as an eMail MIME header.
- 2888 • All other MIME headers that constitute the ebXML Message Envelope MUST also become part of the
 2889 eMail MIME header.
- 2890 • The SOAPAction MIME header field must also be included in the eMail MIME header and MAY have
 2891 the value of ebXML:

2892 SOAPAction: "ebXML"

2893 Where Service and Action are values of the corresponding elements from the ebXML

2894 **MessageHeader.**

- 2895 • The "MIME-Version: 1.0" header must appear as an eMail MIME header.
- 2896 • The eMail header "To:" MUST contain the [RFC822] compliant eMail address of the ebXML Message
 2897 Service Handler.
- 2898 • The eMail header "From:" MUST contain the [RFC822] compliant eMail address of the senders
 2899 ebXML Message Service Handler.
- 2900 • Construct a "Date:" eMail header in accordance with [RFC822]
- 2901 • Other headers MAY occur within the eMail message header in accordance with [RFC822] and
 2902 [RFC2045], however ebXML Message Service Handlers MAY choose to ignore them.

2903 The example below shows a minimal example of an eMail message containing an ebXML Message:

```

2904 From: ebXMLhandler@example.com
2905 To: ebXMLhandler@example2.com
2906 Date: Thu, 08 Feb 2001 19:32:11 CST
2907 MIME-Version: 1.0
2908 SOAPAction: "ebXML"
2909 Content-type: multipart/related; boundary="Boundary"; type="text/xml";
2910 start="<ebxhmheader111@example.com>"
2911
2912 --Boundary
2913 Content-ID: <ebxhmheader111@example.com>
2914 Content-Type: text/xml
2915
2916 <?xml version="1.0" encoding="UTF-8"?>
2917 <SOAP-ENV:Envelope xmlns:SOAP-ENV='http://schemas.xmlsoap.org/soap/envelope/'
2918 xmlns:eb='http://www.ebxml.org/namespaces/messageHeader'>
2919 <SOAP-ENV:Header>
2920 <eb:MessageHeader SOAP-ENV:mustUnderstand="1" eb:version="1.0">
2921 <eb:From>
    
```



```

2923     <eb:PartyId>urn:duns:123456789</eb:PartyId>
2924     </eb:From>
2925     <eb:To>
2926         <eb:PartyId>urn:duns:912345678</eb:PartyId>
2927     </eb:To>
2928     <eb:CPAId>20001209-133003-28572</eb:CPAId>
2929     <eb:ConversationId>20001209-133003-28572</eb:ConversationId>
2930     <eb:Service>urn:services:SupplierOrderProcessing</eb:Service>
2931     <eb:Action>NewOrder</eb:Action>
2932     <eb:MessageData>
2933         <eb:MessageId>20001209-133003-28572@example.com</eb:MessageId>
2934         <eb:Timestamp>2001-02-15T11:12:12Z</Timestamp>
2935     </eb:MessageData>
2936     <eb:QualityOfServiceInfo eb:deliverySemantics="BestEffort" />
2937 </eb:MessageHeader>
2938 </SOAP-ENV:Header>
2939 <SOAP-ENV:Body>
2940     <eb:Manifest SOAP-ENV:mustUnderstand="1" eb:version="1.0">
2941         <eb:Reference xlink:href="cid:ebxmlpayload111@example.com"
2942             xlink:role="XLinkRole"
2943             xlink:type="simple">
2944             <eb:Description xml:lang="en-us">Purchase Order 1</eb:Description>
2945         </eb:Reference>
2946     </eb:Manifest>
2947 </SOAP-ENV:Body>
2948 </SOAP-ENV:Envelope>
2949
2950 --Boundary
2951 Content-ID: <ebxmhheader111@example.com>
2952 Content-Type: text/xml
2953
2954 <?xml version="1.0" encoding="UTF-8"?>
2955 <purchase_order>
2956     <po_number>1</po_number>
2957     <part_number>123</part_number>
2958     <price currency="USD">500.00</price>
2959 </purchase_order>
2960
2961 --Boundary--

```

2962 B.3.3 Response Messages

2963 All ebXML response messages, including errors and acknowledgements, are delivered *asynchronously*
 2964 between ebXML Message Service Handlers. Each response message MUST be constructed in
 2965 accordance with the rules specified in the section titled "Sending ebXML messages over SMTP"
 2966 elsewhere in this document.

2967 ebXML Message Service Handlers MUST be capable of receiving a delivery failure notification message
 2968 sent by an MTA. A MSH that receives a delivery failure notification message SHOULD examine the
 2969 message to determine which ebXML message, sent by the MSH, resulted in a message delivery failure.
 2970 The MSH SHOULD attempt to identify the application responsible for sending the offending message
 2971 causing the failure. The MSH SHOULD attempt to notify the application that a message delivery failure
 2972 has occurred. If the MSH is unable to determine the source of the offending message the MSH
 2973 administrator should be notified.

2974 MSH's which cannot identify a received message as a valid ebXML message or a message delivery
 2975 failure SHOULD retain the unidentified message in a "dead letter" folder.

2976 A MSH SHOULD place an entry in an audit log indicating the disposition of each received message.

2977 **B.3.4 Access Control**

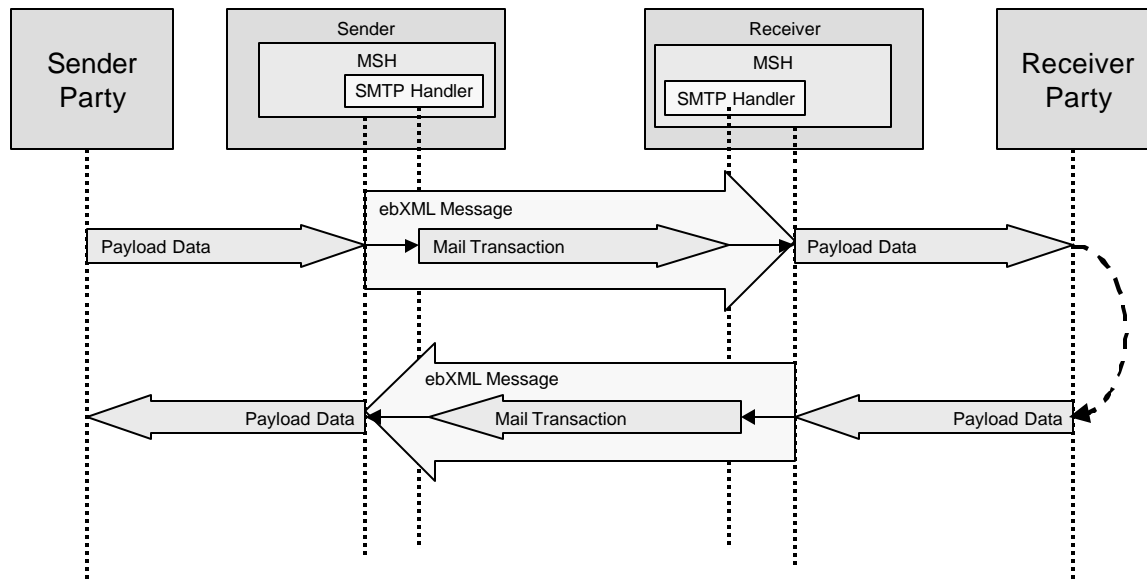
2978 Implementers MAY protect their ebXML Message Service Handlers from unauthorized access through the
 2979 use of an access control mechanism. The SMTP access authentication process described in "SMTP
 2980 Service Extension for Authentication" [RFC2554] defines the ebXML recommended access control
 2981 mechanism to protect a SMTP based ebXML Message Service Handler from unauthorized access.

2982 **B.3.5 Confidentiality and Communication Protocol Level Security**

2983 An ebXML Message Service Handler MAY use transport layer encryption to protect the confidentiality of
 2984 ebXML messages. The IETF "SMTP Service Extension for Secure SMTP over TLS" specification
 2985 [RFC2487] provides the specific technical details and list of allowable options, which may be used.

2986 **B.3.6 SMTP Model**

2987 All *ebXML Message Service* messages carried as mail in a [SMTP] Mail Transaction as shown in the
 2988 figure below.



2989
 2990

2991 **B.4 Communication Errors during Reliable Messaging**

2992 When the Sender or the Receiver detects a transport protocol level error (such as an HTTP, SMTP or
 2993 FTP error) and Reliable Messaging is being used then the appropriate transport recovery handler will
 2994 execute a recovery sequence. Only if the error is unrecoverable, does Reliable Messaging recovery take
 2995 place (see section 10).

2996 **Disclaimer**

2997 The views and specification expressed in this document are those of the authors and are not necessarily
2998 those of their employers. The authors and their employers specifically disclaim responsibility for any
2999 problems arising from correct or incorrect implementation or use of this design.

3000

3001

3002

3003

3004

3005 **Copyright Statement**

3006 Copyright © UN/CEFACT and OASIS, 2001. All Rights Reserved

3007 This document and translations of it MAY be copied and furnished to others, and derivative works that
3008 comment on or otherwise explain it or assist in its implementation MAY be prepared, copied, published
3009 and distributed, in whole or in part, without restriction of any kind, provided that the above copyright notice
3010 and this paragraph are included on all such copies and derivative works. However, this document itself
3011 MAY not be modified in any way, such as by removing the copyright notice or references to the ebXML,
3012 UN/CEFACT, or OASIS, except as required to translate it into languages other than English.

3013 The limited permissions granted above are perpetual and will not be revoked by ebXML or its successors
3014 or assigns.

3015 This document and the information contained herein is provided on an "AS IS" basis and ebXML
3016 disclaims all warranties, express or implied, including but not limited to any warranty that the use of the
3017 information herein will not infringe any rights or any implied warranties of merchantability or fitness for a
3018 particular purpose.